

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Computer Science**

Monitoring and automated regulation of parameters of hydroponic system environment

Ali Akhmadov

**Supervisor: Ing. David Kadleček, Ph. D.
Field of study: Software Engineering and Technology
May 2021**

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Akhmadov** Jméno: **Ali** Osobní číslo: **483815**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Monitoring a automatizované regulace parametrů prostředí hydroponického systému

Název bakalářské práce anglicky:

Monitoring and automated regulation of parameters of hydroponic system environment

Pokyny pro vypracování:

Vytvořte simulaci monitoringu a automatizované regulace parametrů prostředí kolem hydroponického systému.

Monitorované parametry:

- Hladina CO₂
- PAR (photosynthetic active radiation)
- Teplota
- Vlhkost

Automatizace regulace:

- Větrák (in/out) na regulaci CO₂ a teploty
- Bomba s CO₂ pro regulaci hladiny CO₂
- Zvlhčovač
- Přívodní ventil na vodu a odpust'
- Topení
- Chlazení
- Zařívky - řeší se regulace podle typu rostliny a prostředí (out/indoor)

Výstup:

- Navrh komponent a jejich integrace
- Prototyp formou počítačové simulace nebo fyzického nasazení

Seznam doporučené literatury:

- [1] William Texier – Hydroponie pro každého. ISBN: 978-2-84594-161-8
[2] Stuart Russell / Peter Norvig – Artificial Intelligence: A Modern Approach. ISBN: 9781292153964

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Kadleček, Ph.D., Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.02.2021**

Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. David Kadleček, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank Ing. David Kadleček, Ph.D. for being my supervisor. I would also like to thank Ing. Jiří Šebek, Jakub Szasz, Adam Kučera and Pavel Wimmer for the huge support provided during the project.

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, for any other degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

In Prague, May, 2021

Abstract

This thesis aims to design and develop a responsive automated hydroponic system by implementing monitoring and regulation of environmental parameters around the system. In recent decades, hydroponics has become quite popular and a lot of researches have revealed that plants grown hydroponically are of high quality and consume fewer resources than traditional growing methods. Hydroponic systems are affected by several factors, however, this thesis primarily focuses on controlling the light intensity, air temperature and humidity. This work contains the study of the theory around hydroponics, analysis of hardware, software and available technologies, design and implementation of the system and finally, the evaluation of the achieved results.

Keywords: Automated Hydroponic System, Hydroponics, Environmental Parameters

Supervisor: Ing. David Kadleček, Ph. D.
Prague, Technická 2, A3-222

Abstrakt

Cílem této práce je navrhnout a vyvinout responzivní automatizovaný hydroponický systém pomocí implementace monitoringu a regulace parametrů prostředí kolem systému. V posledních desetiletích se hydroponie stala docela populární a řada výzkumů objevila, že rostliny pěstované hydroponicky jsou vysoce kvalitní a spotřebovávají méně zdrojů než tradiční metody pěstování. Hydroponické systémy jsou ovlivňovány několika faktory, avšak tato práce se primárně zaměřuje na řízení světla, teploty vzduchu a vlhkosti. Tato práce obsahuje studium teorie hydroponie, analýzu hardware, software a dostupných technologií, návrh a implementaci systému a nakonec zhodnocení dosažených výsledků.

Klíčová slova: Automatizovaný Hydroponický Systém, Hydroponie, Parametry prostředí

Překlad názvu: Monitoring a automatizované regulace parametrů prostředí hydroponického systému

Contents

1 Introduction	1	5.2 Non-functional Requirements . . .	28
2 Familiarization with the topic	3	5.3 Hydroponic System Environment Structure	28
2.1 Introduction to Hydroponics	3	5.4 System Component Diagram . . .	29
2.2 Pros and Cons of Hydroponics . . .	4	5.5 System Deployment Diagram . . .	30
2.2.1 Advantages	4	5.6 Communication Between Services	31
2.2.2 Disadvantages	4	5.6.1 Publish-Subscribe Pattern . . .	31
2.3 Hydroponic Systems	5	5.6.2 Redis Pub/Sub Channels . . .	32
2.3.1 Ebb and Flow	5	6 Implementation	33
2.3.2 Deep Water Culture	6	6.1 Arduino Uno Sketches	33
2.3.3 Nutrient Film Technique	6	6.1.1 Sensors Sketch	33
2.3.4 Dutch Bucket	7	6.1.2 Actuators Sketch	36
2.4 Growing Media	8	6.2 Redis Persistence Service	39
2.4.1 Coconut Coir	8	6.2.1 Sensors and Actuators Configuration Data	40
2.4.2 Perlite	8	6.2.2 Serial Connections Management	40
2.4.3 Rockwool	9	6.2.3 Persisting Sensor Data	41
2.5 Environmental Parameters	9	6.3 AWS Persistence Service	42
2.5.1 Air	9	6.3.1 Persisting Sensor Data	43
2.5.2 Grow Lights	10	6.3.2 Managing Data Corruption . .	43
2.5.3 Water Solution	10	6.4 Data Processing and Controlling Service	44
3 Hardware Analysis	13	6.4.1 Regulation Rules and Coefficients	45
3.1 Computing Power	13	6.4.2 Environmental Parameters Regulation	47
3.1.1 Raspberry Pi	13	7 System Evaluation	49
3.1.2 Arduino Uno	14	7.1 Sensor Data Measurement and Persistence	49
3.1.3 Raspberry Pi vs Arduino . . .	15	7.2 Parameter Regulation Outputs .	50
3.2 Sensors	16	7.2.1 Air Temperature	50
3.3 Actuators	17	7.2.2 Lights	53
4 Software and Technologies Analysis	19	7.2.3 Humidity	53
4.1 Communication Protocols	19	8 Conclusion	55
4.1.1 Inter-Integrated Circuit	20	A Source Code	57
4.1.2 Universal Asynchronous Receiver and Transmitter	20	B Installed System Images	59
4.1.3 Serial Peripheral Interface . .	21	C List of Abbreviations	61
4.2 Software Architecture	21	D Bibliography	63
4.2.1 Monolithic Architecture	21		
4.2.2 Microservices Architecture . .	22		
4.2.3 Comparison Table	23		
4.3 Data Storage	23		
4.4 Programming Languages	25		
4.4.1 Arduino Programming Language	25		
4.4.2 Python	26		
5 System Design and Requirements	27		
5.1 Functional Requirements	27		

Figures

2.1 Ebb and Flow Schema	5
2.2 Deep Water Culture Schema	6
2.3 Nutrient Film Technique Schema	7
2.4 Dutch Bucket Schema	7
2.5 Coconut Coir	8
2.6 Perlite	8
2.7 Rockwool	9
3.1 Raspberry Pi	14
3.2 Arduino Uno	14
4.1 Monolithic Architecture	22
4.2 Microservices Architecture	22
5.1 System Environment Schema	28
5.2 System UML Component Diagram	29
5.3 System UML Deployment Diagram	30
5.4 Redis Pub/Sub Channels	32
6.1 Sensor Handling UML Sequence Diagram	34
6.2 Actuators Handling UML Sequence Diagram	36
6.3 Redis Persistence Service UML Sequence Diagram	39
6.4 AWS Persistence Service UML Sequence Diagram	42
6.5 Data Processing and Controlling Service UML Sequence Diagram	45
7.1 AWS DynamoDB Sensor Data	50
7.2 Low Room and High Server Room Air Temperature Logs	51
7.3 Low Room and Low Server Room Air Temperature Logs	52
7.4 High Room and Low Server Room Air Temperature Logs	52
7.5 High Room and High Server Room Air Temperature Logs	53
7.6 PAR Regulation Logs	53
7.7 Low Humidity Logs	54
7.8 High Humidity Logs	54
B.1 Growing Plants	59
B.2 Hardware Box	60

Tables

3.1 Raspberry Pi vs Arduino Uno	15
3.2 List of Sensors	16
3.3 List of Actuators	17
4.1 Comparison of Communication Protocols	19
4.2 Comparison of Software Architectures	23
4.3 Comparison of SQL and NoSQL databases	24
5.1 Functional Requirements	27



Chapter 1

Introduction

Agriculture on a big scale is, and will always be, an integral part of society. As the world population continues to grow, so must the food production. With the increased focus on environmental sustainability, the option of growing plants in non-native environments is gaining more and more interest from the public. Conventional gardening has a lot of disadvantages that can be addressed by hydroponics.

Hydroponic systems use water as a growing medium instead of soil and lead to great results. The system produces higher yields and can be designed to support continuous and effective production throughout the year. Creating a self-sufficient hydroponic system operating without dependence on outside climate enables the possibility of growing plants in any part of the globe.

The goal of this thesis is to construct a reliable automated hydroponic system by implementing monitoring and regulation of environmental parameters around the system. I have chosen this topic because I think that automated hydroponic systems can provide an effective solution to the growing consumption of food and minimize the labor required to grow and maintain plants. The system that I have implemented is part of the larger project, which also includes the regulation of water-related physical parameters. That part of the system was managed by my colleague and both our solutions are integrated into one hydroponic automated system. [1]

In the following chapters, I will describe the main aspects of hydroponics and the theory around it. After that, I will go through the steps I had to take to implement the system and evaluate the results.

Chapter 2

Familiarization with the topic

In this chapter, we will get familiar with the general parts and terms of hydroponics that it is useful to know while implementing an automated hydroponic gardening system. Mainly I will focus on basic hydroponic system types, substrates that are used to maintain plants and environmental parameters that are being measured in the scope of this project as well as nutrient solution parameters to have a basic understanding of the entire system.

2.1 Introduction to Hydroponics

The word hydroponics is derived from the combination of two Greek words, *hydro* meaning water and *ponos* meaning labor. The word first appeared in a scientific magazine article published in 1937 and authored by W.F. Gericke. Dr. Gericke started experimenting with hydroponic techniques in the late 1920s and then published one of the early books on soilless gardening.

There are lots of definitions for the word *hydroponics* in various sources and each has some small difference. However, the most common aspect of all definitions is that hydroponics means growing plants without utilizing soil, with the sources of nutrients either from a nutrient solution or nutrient-enriched water. [2]

It turns out that growing plants in nutrient-rich water has been practiced for centuries. For instance, the ancient Hanging Gardens of Babylon and the floating gardens of the Aztecs were hydroponic in nature. The basic concepts for the hydroponic growing of plants were established in the 1800s. The culture of cultivating plants in a soilless environment was then popularized in the 1930s in a series of publications by a California scientist (Gericke). Since then, hydroponics started to develop its potential. During World War II, the U.S. Army constructed large hydroponic gardens on several islands in the western Pacific to supply vegetables to troops. Since the 1980s, the hydroponic technique has gained considerable commercial value for vegetable and flower production and in 1995 there were over 60,000 acres of hydroponically grown vegetables throughout the world. The popularity of hydroponics continued to grow and is still growing nowadays. [2]

■ 2.2 Pros and Cons of Hydroponics

In this section, I will analyze the advantages and disadvantages of hydroponics in general. The assumptions and knowledge presented in this section is an aggregation of information available on the internet and my personal beliefs.

■ 2.2.1 Advantages

■ Improved growth and yield

In the majority of cases, hydroponic systems result in faster-growing and higher-yielding plants. This is due to the increased concentration of nutrients in the nutrient solution and the carefully controlled environmental parameters. [3]

■ An extended growing season

In the outdoor environment, plants are tied to the environment they grow in and cannot produce a yield in some periods of the year. With a hydroponic system, plants can grow all year round because the grower controls the environmental parameters. [4]

■ Less water consumption

Besides the fact that hydroponics is primarily based on using water to grow plants, it uses between 80 to 90% less water than plants cultivated in the traditional manner. In classical gardening, a large amount of water is applied to the soil to allow adequate moisture to reach the root zone. While water travels to the roots, it evaporates and only a small percentage of it reaches the roots. [3]

■ Plants can grow everywhere

Traditional gardening requires outdoor space for plants, while hydroponic systems can be easily incorporated into many homes. Moreover, in an isolated environment, parameters can be more effectively adjusted for each corresponding plant.

■ 2.2.2 Disadvantages

■ Expensive to set up

Hydroponic systems are more expensive to acquire and construct. It requires a lot of components, costs of which range depending on the type and size of the building system.

■ Constant monitoring and maintenance

Hydroponic systems should be frequently monitored compared to traditional gardening. All system components need to be observed and adjusted if required - lights, temperature, nutrient solution and many other aspects. With poor control over the system, plants will result in low yield and can obtain diseases.

■ 2.3 Hydroponic Systems

The first step to starting hydroponic gardening is choosing a suitable system. The systems are distinguished as active or passive. By active is meant that the nutrient solutions are moved, usually by a pump, whereas passive means that a wick or the anchor of the growing medium helps flow the nutrients to the roots of plants [5]. There are hundreds of variations of hydroponic systems available for use, however, there are only several fundamental types of hydroponic systems, on which all variations are based. In this section, I will describe some of the most popular hydroponic systems, on which our solution can be built.

■ 2.3.1 Ebb and Flow

The Ebb and Flow system represented in figure 2.1 is characterized by an automatic flood and drain watering technique, in which plants are flooded temporarily and regularly. The water or nutrient solution in the reservoir ascends to a growth tray via a water pump, accumulates to a certain level, and stays in the grow tray for a prescribed amount of time providing water and nutrients to the plants. After the expiration of time, the solution is drained back into the reservoir through a tubing system. To maintain this circulation system, continuous observation should be present to control the amount of water provided to the system. Although it is possible to grow a lot of different kinds of plants and provide them with a large amount of water, root diseases and growth of algae or molds may occur in this system. [5]

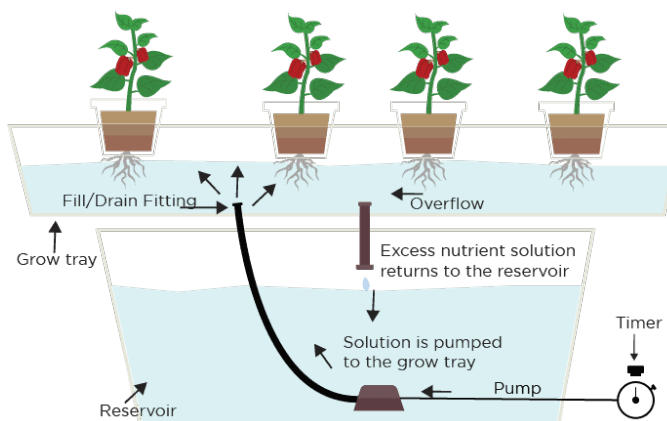


Figure 2.1: Ebb and Flow Schema

2.3.2 Deep Water Culture

The Water Culture System given in figure 2.2 is a simple model, composed of a reservoir, a tubing system, an airstone, an air pump, and a floating platform. This system was developed so that plants can be grown with roots constantly suspended in water. This contributes to active food production for plants: a floating platform holds plants or pots in a reservoir, where the root parts are constantly immersed in the water or nutrient solution, and oxygen is supplied by an airstone and air pump. To optimize growing conditions, it is necessary to monitor the oxygen level and nutrient concentrations, salinity, and pH. Although this environment suits many different plants, especially cucumber and radish, large or long-term crops may not grow well. Moreover, algae and molds can grow rapidly in the reservoir. [5]

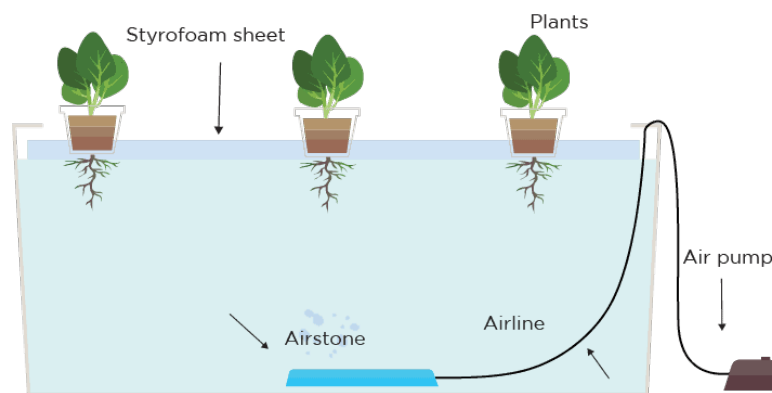


Figure 2.2: Deep Water Culture Schema

2.3.3 Nutrient Film Technique

The nutrient film technique (NFT) described in figure 2.3 is a system generated to compensate for the weak points of the Ebb and Flow systems. With the NFT technique, the plants are grown in channels, via which the nutrient solution is pumped. The roots are kept moist by the thin film of nutrient solution as it passes by. The ideal way is to expose the bottom of the roots to the nutrient solution, while the top is kept moist but not water-logged. [5] Most NFT channels are fed continuously at a rate of approximately 1 liter per minute. Since the plant roots are not maintained in a growing medium, it is obligatory to keep them moist at all times. In the majority of NFT systems, the nutrient solutions mixed beforehand in a primary reservoir are cycled through the channels and back to the reservoir. The nutrient reservoir can be automatically regulated, and with proper aeration and pH adjustment can effortlessly be kept fresh for weeks. [6]

NFT is ideal for short-term crops, however, for long-term crops, e.g., cucumber and tomatoes, larger NFT channels can be applied. One of the great advantages of the NFT system is that the crops are clean and no washing is necessary. [6]

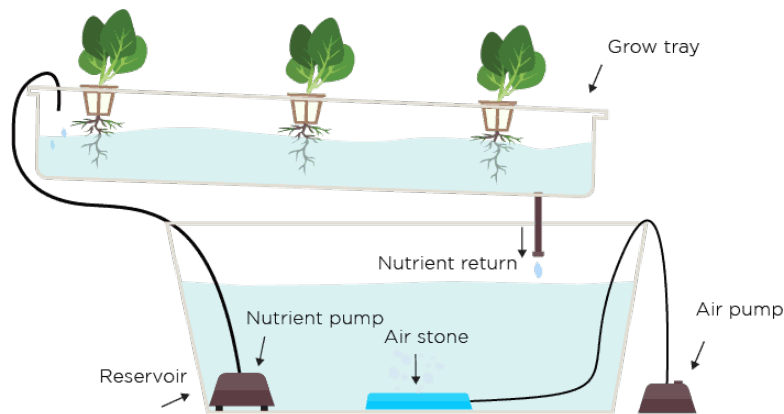


Figure 2.3: Nutrient Film Technique Schema

2.3.4 Dutch Bucket

As the name implies, Dutch Bucket illustrated in figure 2.4 was first introduced in Holland and is now effectively used by commercial growers for different kinds of plants. The Dutch Bucket method allows the grower to use any growing medium including coco-coir, perlite, gravel, LECA stone and sand. The Dutch Bucket system is represented by a 2.5 gallon (9.5 liters) bucket with a special drain that maintains a small reserve of nutrients at the bottom as a precautionary measure. Each bucket is fed a nutrient solution by a single or double dripper, and it drains through the bucket into a common drain tube made from a 1.5 inch (3.81 cm) PVC pipe. The reservoir is placed below the level of the drainpipe, and with the help of gravity, the solution is carried back to it. Subsequently, the pump re-circulates the nutrients back to the drippers to start the cycle over again. The method is mostly suited for large, long-term crops, e.g., cucumbers, vine tomatoes and roses. [7] It was decided that Dutch Bucket method will be used in the scope of this project due to its universality, efficiency and simplicity.

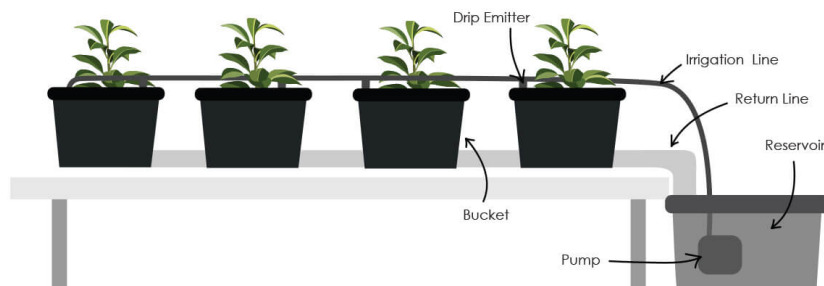


Figure 2.4: Dutch Bucket Schema

■ 2.4 Growing Media

As hydroponic implies a soilless environment, plants need something that would hold and support them. Moreover, the media should be able to transmit moisture, nutrients and oxygen to the roots of the plants. A growing medium is responsible for this job. There are lots of variations of media but only a few are considered the best.

■ 2.4.1 Coconut Coir

Coconut coir, shown in figure 2.5, is an organic material created from coconut shell husks. This type of media is able to hold water and air well and is environmentally friendly. Nevertheless, the media does not have good drainage; thus, it often mixes with other materials and becomes uncompressed after several uses. [8]



Figure 2.5: Coconut Coir

■ 2.4.2 Perlite

Perlite is one of the commonly used media in hydroponics. It is created by expanding volcanic glass under extremely high temperatures. Consequently, countless small white particles pop out. This medium is distinguished by high oxygen retention and its reusability. However, it can be too lightweight for some systems and produce excess dust from particles. Perlite is shown in figure 2.6. [8]



Figure 2.6: Perlite

2.4.3 Rockwool

Rockwool has been used widely both by hobbyists and commercial farmers in recent decades. This material is generated by melting rocks and spinning them into bundles of filament fibers. Rockwool is a versatile inert growing medium that is used in many different hydroponic systems, especially recirculation types. The great pros of rockwool are that it holds water very well, provides good oxygen retention and has a variety of sizes and shapes. On the other hand, rockwool is not pH neutral and produces dust from particles that can harm the plants. Rockwool is shown in figure 2.7. [8]



Figure 2.7: Rockwool

2.5 Environmental Parameters

In this part, I will go through the environmental parameters that will be measured in this project solution. However, I will also slightly cover water-related parameters as it is good to have an idea of the overall system architecture.

2.5.1 Air

Air is a vital component that plants need to live and grow healthy and efficiently. Plants highly depend on air parameters, such as temperature, humidity and carbon dioxide.

Temperature

Plants are constantly processing energy 24 hours a day. The speed, at which plants process energy, is directly related to air temperature. The best temperature for warm-season crops is 16°C at night and 24°C during the daytime. On the other hand, cool-season crops grow better at 10°C at nighttime and 16°C during the day. Of course, these values are not the same for all kinds of plants. Specific optimal minimum and maximum values depend on species. If the temperature is too low, plant growth will be slower and some purpling of the leaves may arise. Nevertheless, too high temperatures may result in poor-quality plants. [9]

■ Humidity

Humidity is one of the major factors influencing the process of photosynthesis and the entire growing process. Humidity is the concentration of water vapor present in the air. Maintaining optimal humidity value is often a challenge because it depends on other factors such as air temperature.

Warm air maintains more water than cold air. It is important to understand that the percentage of humidity is related to the water vapor, which air can maintain at the given temperature. For instance, in the room with 10°C temperature and 100% of relative humidity, the concentration of water vapor is half as much as in the room with 20°C temperature and 100% of relative humidity. This indicates that increasing temperature will lead to humidity reduction. Oppositely, decreasing temperature will make humidity level increase. [3]

In the humid environment plant leaves grow larger. Conducted experiments have revealed that maximal growth rate is reached at the humidity level ranging from 60% to 80%. However, in order to avoid extremes, it is better to keep humidity level in range 65 - 75%. Humidity level is adjusted using humidifiers and ventilators synchronized together. [3]

■ CO2

Plants are the only organisms able to feed on sunlight. During the daytime, plants consume CO₂ for photosynthesis and release oxygen. While growing indoors, it is important to maintain natural airflow. In order to bring new CO₂ to the grow room for plants, a proper air renewal system should be implemented. Usually, this is solved by ventilators, which simulate natural air conditions for plants. The power and number of ventilators depend on the size of the space where plants grow. [3]

■ 2.5.2 Grow Lights

In order to survive, plants need food. They use sunlight to make food (sugars) via a process called photosynthesis. In modern hydroponics, artificial lights are used to provide plants with a comfortable environment. There are different units of measurement for lighting but the one that is measured in hydroponics is PAR (Photosynthetically Active Radiation). Nowadays one of the widely used sources of artificial light in hydroponics is LED (Light Emitting Diode). LED has become so popular due to its efficiency and lightweightness. They produce much light with little electricity. LEDs are made up of many diodes and growers can customize which light colors and light wavelength they need for different species. [8]

■ 2.5.3 Water Solution

In hydroponics, water is the main element providing plants with nutrients. In most cases water solution is maintained in a reservoir and is periodically

transported to the plant roots. The amount of water and the volume of the reservoir depend on the type of hydroponic system as well as the species and quantity of plants.

To let plants grow effectively, it is crucial to maintain optimal values for water parameters, mainly water temperature, pH (potential of hydrogen) and EC (Electrical Conductivity). The values differ depending on the type and growth phase of a plant.

■ Temperature

Temperature of water solution plays a critical role in the process of growing plants. It determines an amount of oxygen in a solution. The higher the temperature, the lower the concentration of oxygen in the solution. At the same time, higher temperatures contribute to metabolism of plants. To reach a good process of nutrients transportation to plants, there has to be a relatively high amount of oxygen. Water temperature also has a great impact on the speed of plant growth. The ideal interval for temperature ranges from 18°C to 24°C. This is the optimal value, at which balanced growth can be maintained. [3]

■ Potential of Hydrogen

pH is considered to be the most important factor in hydroponics. pH level should be measured and regulated regularly, so that the nutrients are evenly and effectively distributed between the roots. It is clear that the optimal pH values vary from plant to plant, however, it is considered that the best pH for hydroponics is a range of 5.5-6.5. [3]

■ Electrical Conductivity

EC is the measurement of electrical conductivity within nutrient solution. It indicates the amount of available nutrients in water solution. Once minerals are added into water, the dissolved salts allow it to conduct electricity. The higher the concentration of salts, the higher the electrical conductivity. For most plants optimal EC values range from 1.2 to 1.6 during the vegetative stage and 1.6-2.4 during flowering. However, these values depend on the type of the growing plant. [10]

Chapter 3

Hardware Analysis

This chapter describes the hardware part of the project by giving information about the computing power components, sensors and actuators that will appear in the implementation.

3.1 Computing Power

In this section, I will go through computing power components, i.e. Raspberry Pi and Arduino Uno. These two were chosen due to the fact that they have large communities, documentation and, in general, they are the most popular components used in such kinds of projects. Another advantage is that they are pretty cheap and simple to use.

3.1.1 Raspberry Pi

Raspberry Pi, also abbreviated as RPI, is a small, powerful, cheap and hackable computer board that can be held on palm with the capacity of functioning as a full-fledged computer. Using RPI, one can design and implement different applications and prototype models with the minimum knowledge of programming. The great advantage of Raspberry Pi is that all the components of a computer are integrated into a single chip that comprise CPU, memory, I/O port, secondary memory and other components. Some versions of RPI have in-built Bluetooth and WiFi modules along with Gigabit Ethernet for data transfer and connectivity. Via the USB ports, one can connect peripheral devices, e.g. keyboard and mouse. RPI can be used as a strong device in controlling applications and appliances with the internet; therefore, it represents an ideal platform for IoT based applications. [11]

In this project, Raspberry Pi 4 Model B will be used as home for complex calculations on gathered data and will provide connectivity with the internet. This version of RPI is the best as it has WiFi and Ethernet modules, Bluetooth and USB connectors. Using a serial network, It will be connected to Arduino Uno, which will be responsible for data collection and activation of actuators according to the instructions from RPI. Raspberry Pi is shown in figure 3.1.

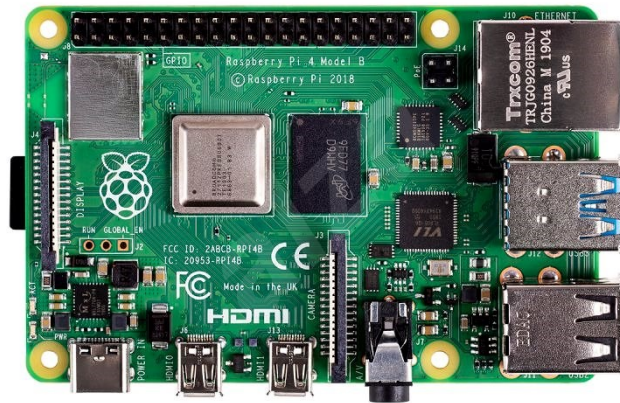


Figure 3.1: Raspberry Pi

3.1.2 Arduino Uno

The Arduino Uno is a microcontroller board developed by Arduino.cc. The board contains sets of digital and analog I/O pins that may be interfaced to various expansion boards and other circuits. Arduino Uno has 14 digital I/O pins, six of which are capable of PWM output, 6 analog I/O pins and can be programmed with the Arduino IDE (Integrated Development Environment) via a type B cable. Arduino boards are capable of reading inputs - lights on a sensor, touching a button, or sending and receiving a message - and turning it into an output - activating a motor, lighting an LED, etc. To achieve this, Arduino uses its own programming language with Arduino IDE based on processing. [12]

In this project, Original Arduino Uno will be used to collect data from sensors and to activate actuators. Collected data will be sent to Raspberry Pi, which will save data to the database, make computations on the data and send back instructions to Arduino. Arduino Uno is illustrated in figure 3.2.

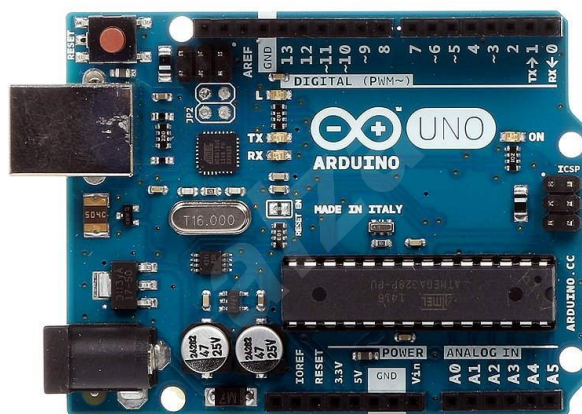


Figure 3.2: Arduino Uno

3.1.3 Raspberry Pi vs Arduino

Raspberry Pi and Arduino are quite different boards. The main obvious difference is that Arduino is a microcontroller while RPI is a mini computer. Each one has its own pros and cons, and it is not a trivial task to choose the best one. The following table 3.1 describes some of the main differences between them. [13]

	Raspberry Pi	Arduino Uno
System	Single Board Computer	Microcontroller
Operating System	Run on an operating system	No operating system
Program execution	Can run multiple programs simultaneously	Runs only one program again and again
Connection to Internet	Can be easily connected to the internet using Ethernet port and USB Wi-Fi dongles	Arduino requires external hardware to connect to the internet and this hardware is addressed properly using code
Interfacing Sensors	Requires complex tasks like installing libraries and software for interfacing sensors and other components	Very simple to interface sensors and other electronic components
Storage	Does not have storage on board. Provides an SD card port	Can provide onboard storage
Advantages	Highly flexible with different operating systems and able to run several applications simultaneously	Perfect for reading sensor values and controlling actuators
Disadvantages	More expensive and cannot read analog sensor values	Limited usage due to lack of computation power and no OS
Price	Relatively expensive	Low cost

Table 3.1: Raspberry Pi vs Arduino Uno

For the purpose of this project, both of the boards will be present in the solution. As can be seen in the comparison table, Arduino is better for working with sensors and actuators while RPI shows great results in computation and connectivity to the internet. Thus, the final solution will include Arduino, which will be connected to sensors, actuators and Raspberry Pi via a serial network. RPI will be responsible for connectivity to the internet and complex computations on collected data from sensors sent by Arduino and will send instructions back to Arduino to activate relevant actuators.

3.2 Sensors

In this section, I will briefly introduce the sensors that were chosen to measure the environmental parameters. The sensors are divided according to environmental parameters they are able to measure. Each sensor has a library support for Arduino Uno, which enables to manipulate them easily. Sensors represented in table 3.2 were chosen according to their efficiency, performance and measuring accuracy.



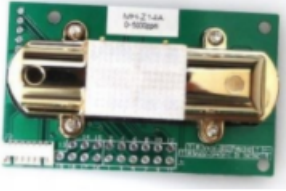
Sensor Name	Photo	Measured Parameter
SHT20		Temperature and Humidity
PAR Sensor		PAR (Photosynthetically Active Radiation)
MH-Z14		Carbon Dioxide

Table 3.2: List of Sensors

3.3 Actuators

In this section, I will describe the actuators given in table 3.3 that were chosen to regulate the environmental parameters. Similarly to sensors, actuators were chosen according to their performance and efficiency. Each actuator can be conveniently controlled via Arduino Uno using third party libraries or standard Arduino functions, such as producing output signal for a certain pin or generating PWM (Pulse-width modulation) signals.

Actuator name	Photo	Goal
LED		Produces light
Fan		Circulates the air in the room to regulate CO2 levels and air temperature
Heater		Raises air temperature in the room
Humidifier		Regulates air humidity levels

Table 3.3: List of Actuators

Chapter 4

Software and Technologies Analysis

Implementing an automated hydroponic gardening system requiring lots of hardware and software configurations and interconnections can be a hard task to accomplish. To achieve this goal, one should thoroughly analyze the technologies that can be used for realization. In this chapter, I will scrutinize existing communication protocols, data storage options, potential software architectures and programming languages that can be applied in the final solution to the problem.

4.1 Communication Protocols

To create automated and efficient environment for hydroponic system, nodes of the system require communication mechanisms in order to exchange data retrieved from sensors to the aggregation node, which in the case of this project is represented by Raspberry Pi, and also to get data from the aggregation node to the sensor node represented by Arduino Uno. With the increased use of Sensor Networks and applications in the most diverse environments, the need for both wired and wireless protocols is growing. Due to the fact that wired protocols are more reliable, secure and can transfer data at higher rates, they are still widely used. The following table 4.1 describes the features of some of the most popular wired and wireless protocols. [14]

Feature	SPI	I ² C	UART	WiFi	Bluetooth	ZigBee
Based Data Rate [Mbps]	20	0.1	0.02 - 10	11	1	0.25
Frequency [GHz]	-	-	-	2.4	2.4	2.4
Nodes/Masters	3	1024	256	32	7	65540
Power Consumption [mA]	-	-	-	100 - 350	1 - 35	1 - 10
Complexity	Medium	Low	Low	High	Medium	Medium

Table 4.1: Comparison of Communication Protocols

As the system must be highly reliable, it was decided to use wired communication protocols to overcome problems, e.g., loss of internet connection. The most common wired technologies are Serial Communication or Universal Asynchronous Receiver and Transmitter (UART), mainly in the form of USB or RS232, SPI and I²C.

■ 4.1.1 Inter-Integrated Circuit

I²C is a serial communication protocol specifically designed for microcontrollers. With I²C, you can connect multiple slaves to a single master and you can have multiple masters controlling single or multiple slaves. It is incredibly popular with modules and sensors, making it useful for projects that require many parts working together. [14]

■ Advantages

- Widely supported
- Easy to connect
- Automatically configured
- Low Power Consumption

■ Disadvantages

- Does not support long distance communication
- Number of nodes is limited
- Does not support high speed connections

■ 4.1.2 Universal Asynchronous Receiver and Transmitter

UART, standing for Universal Asynchronous Receiver and Transmitter, is a simple communication protocol that can allow Arduino to communicate with serial devices. The UART system communicates with digital pin 0 (RX), digital pin 1 (TX) and with another computer via the USB port. [14]

This peripheral on Arduino boards allows direct communication with a computer, in our case Raspberry Pi, thanks to the fact that the Arduino has onboard USB-to-Serial converter.

■ Advantages

- Widely supported
- Robust to errors
- No clock signal needed

■ Disadvantages

- Limited size of 9 bits

■ 4.1.3 Serial Peripheral Interface

SPI is a different form of serial communication protocol specifically designed for microcontrollers to talk to each other. The most notable difference from I²C is that, while you can use multiple masters and slaves with I²C, SPI allows a single master device with a maximum of four slave devices, e.g., sensors and actuators. SPI is commonly used in places where speed is important, such as with SD cards and display modules. [15]

■ Advantages

- Very simple hardware interfacing
- Not limited to any maximum clock speed enabling potentially high speed

■ Disadvantages

- It supports only one master device
- SPI usually requires separate SS (slave select) lines to each slave, which can be problematic if numerous slaves are needed

After scrutinizing all the mentioned characteristics, the protocol stack for this project will be built from I²C and UART. All the sensors and actuators will be connected to Arduino Uno and use I²C for communication, while Arduino itself will be connected to Raspberry Pi using UART serial link.

■ 4.2 Software Architecture

To design a reliable, scalable and effective system, it is extremely important to choose and design right software architecture. Designing software architecture is about arranging components of a system to best fit the desired quality attributes of the system. In this section, I will describe software architectures that can find application in the implementation.

■ 4.2.1 Monolithic Architecture

The monolithic architecture implies that different components of the system are combined into a single unit on a single platform. In most cases, monolithic applications consist of a database, client-side user interface and server-side application. All the software parts are unified and all its functions are managed in one place. The monolithic architecture structure is displayed in figure 4.1. [16]

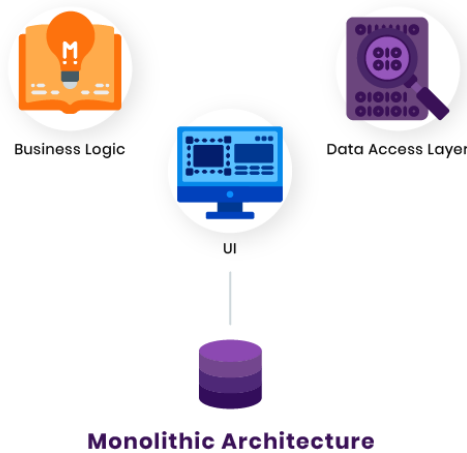


Figure 4.1: Monolithic Architecture

4.2.2 Microservices Architecture

Microservices presented in figure 4.2 is a type of Service Oriented Architecture that focuses on building systems comprised from a series of autonomous components called services. Compared to monolithic architecture, microservice applications consist of multiple independent components that communicate with each other using API to accomplish the goal.

The microservices approach focuses mainly on business priorities, while the monolithic approach is organized around technology layers, UIs, and databases. The microservices approach has become a trend in recent years as more and more enterprises become agile and move toward DevOps. [16]

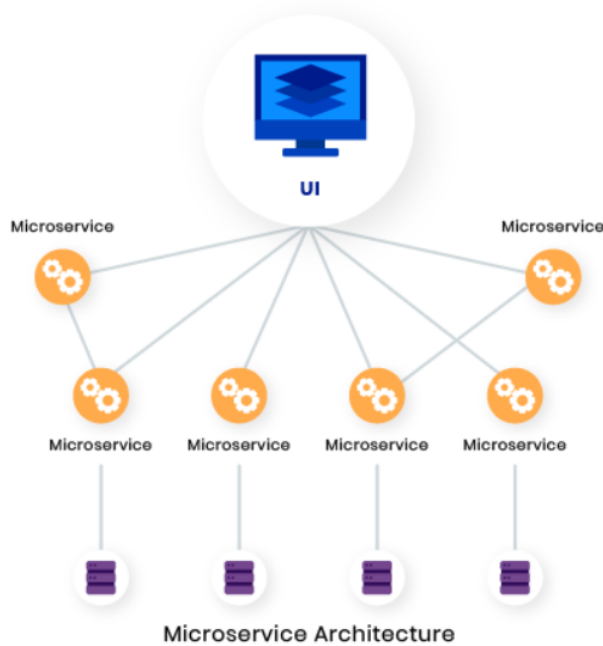


Figure 4.2: Microservices Architecture

4.2.3 Comparison Table

The information presented in table 4.2 compares monolithic and microservices architectures and describes some features of each architecture. [17]

Key	Monolithic architecture	Microservices architecture
Basic	Build as one system and is usually one code-base.	Build as small independent components based on business functionality.
Scale	Difficult to scale and has certain limitations, e.g., hardware.	Very easy to scale based on demand.
Database	It has a shared database.	Each component usually has their own database.
Deployment	Large code base makes IDE slow and build time increases.	Each project is independent and small. So overall development time decreases.
Tightly Coupled and Loosely coupled	Extremely difficult to alter technology, programming language or framework because everything is loosely coupled.	Easy to change technology or framework because every component is independent and can have different used technologies.

Table 4.2: Comparison of Software Architectures

In the scope of this project, described software architectures can be integrated with Raspberry Pi because it is the center for most software part. We have to take into account all the requirements to choose the most suitable one. I decided to use microservices architecture and decompose the system into 3 logical units and implement the communication mechanism between them using Redis Pub/Sub messaging system. More information about individual services, messaging system and overall implementation is described in chapter Implementation.

4.3 Data Storage

To make the hydroponic system consistent and reliable, it is important to equip the system with a data storage mechanism. For this purpose, the database should be very dynamic, rapid, flexible and scalable. When it comes to choosing a modern database, one of the biggest decisions is picking a relational SQL or non-relational NoSQL database. The following table 4.3 provides feature comparisons between relational SQL databases and NoSQL databases. [18]

	SQL	NoSQL
Database Type	One SQL DBMS product	Four general types: key-value, document, wide-column, graph
Schema	Based on foreign key relationships between tables in a database schema. Requires strict definition of schema and data types before inserting data.	Dynamic schema. Does not require schema definition in advance. Different data can be stored together as required. Allows simple schema modification with no downtime.
Data Models	Data records are stored as rows and columns in tables joined via relationships. Contain explicitly defined data types of columns to store a certain piece of data	Supports all types of data - structured, semi-structured, and unstructured.
Scaling Model	Vertical	Horizontal
Data Manipulation	Structured Query Language (SQL) - DML statements are used to manipulate data	Query data efficiently. Object-Oriented APIs used for data manipulation
Software	Oracle, PostgreSQL, MySQL	MongoDB, Redis, Cassandra, DynamoDB, Neo4j, Elasticsearch

Table 4.3: Comparison of SQL and NoSQL databases

Despite the fact that relational database management systems play an important role when processing structured and highly uniform data sets, it is more suitable for collecting data generated from a vast number of enterprise IT systems and where these data are managed in a relatively isolated manner. When it comes to managing more heterogeneous data generated by a number of sensors, devices and gateways, each potentially with their own data structures, databases will require new levels of agility, flexibility and scalability. In the scope of this project, NoSQL databases are proving their value and thus will be applied in the implementation. [18]

From NoSQL databases, DynamoDB is the best candidate to be in the implementation of the system. Another candidate could be MongoDB but it is less scalable and does not have such low latency as DynamoDB [20]. DynamoDB is a fully managed NoSQL database service providing fast and predictable performance. Moreover, it provides a very good scalability. With DynamoDB, one can create a database that can store and retrieve any amount of data and operate at any level of request traffic. It can scale up or scale down throughput capacity without performance degradation. [19]

According to the requirement that the system should operate without internet connection, we also need to have a local storage for data from sensors on Raspberry Pi, where data will be kept intact even in case of internet unavailability. For this purpose, we can pick from several options, such as Redis, MongoDB and Cassandra. Due to the fact that Redis is much faster and

provides publish-subscribe pattern, which is a great and simple option to implement communication between our services, I decided to choose it for storing sensor data locally on Raspberry Pi. Furthermore, Redis has in-built data structures, e.g., list, sorted set and hash, that can be effectively used to manipulate sensor data. [21]

■ 4.4 Programming Languages

One of the major steps to take to implement any kind of software system is choosing programming language or languages. There is a plenty of open source available languages, each usually having its own distinctive feature. However, our language stack highly depends on the hardware we chose to implement an automated hydroponic system. For instance, Arduino Uno is a microcontroller, which can be programmed only by writing sketches in the Arduino Integrated Development Environment in the language similar to the C/C++ language [22]. Nevertheless programs on Raspberry Pi can be written in nearly any language. In this section, I will go through the available programming languages, review their pros and cons and highlight those that will be present in the final solution.

■ 4.4.1 Arduino Programming Language

As it was already mentioned, Arduino Uno boards can only be programmed using Arduino programming language. This means that there is no need to make further investigations for this hardware component in terms of programming languages; thus, Arduino programming language will definitely appear in the final solution. Arduino provides a cross-platform integrated development environment based in Java, which contains multiple code examples, a debug serial console, and is open source. With the Arduino IDE, you can create programs called *sketches*, which are then uploaded to Arduino boards. The syntax is quite similar to C++ and the language itself represents a simplified version of C++ programming language. Arduino has lots of libraries that can be used to communicate with sensors, actuators and Raspberry Pi via serial link. [22]

Each Arduino program must contain at least two functions, which are:

- *setup()* - called once when the program starts. This function will be used for configuration purposes.
- *loop()* - called repetitively as long as Arduino has power. The main logic will be derived from this function, which includes reading values from sensors using the *Wire* library and sending them to Raspberry Pi over serial link. [23]

To communicate with Raspberry Pi via UART, Arduino provides Serial library, which has 2 basic functions:

- *Serial.read()* - reads one byte from UART
- *Serial.write()* - writes one byte into UART

Furthermore, Arduino provides a *Wire* library allowing it to communicate with I²C devices. [24]

■ 4.4.2 Python

Python is a powerful and easy to use programming language that is extremely popular in the modern community. Due to the fact that Python is classified as a high level language, it allows to use fewer lines of code for complex tasks. One of the best things about working with Python on the Raspberry Pi is that Python is a first-class citizen on the platform. The Raspberry Pi Foundation specifically selected Python as the main language because of its power, versatility, and ease of use. Python comes preinstalled on Raspbian, so there is no need to install it manually [25]. There is also a whole set of Python modules that make it easy to work with Raspberry Pi and its peripherals. For example, *serial* from the *PySerial* module will be used in this project to communicate with Arduino Uno over the serial link. [26]

Python is also known as an AI language and has showed great results in the area of machine learning. This is also a great advantage for this project, as it is likely to be incorporated into the AI environment in the further development.

Considering all of the advantages of Python specifically for this project, I decided to use it to implement the software on Raspberry Pi. Alternatives could be Java or C/C++ languages, however, they require more configurations and are sometimes frustrating to use while implementing such types of systems.

Chapter 5

System Design and Requirements

In this chapter, I will go through the main system components, describe system requirements and illustrate the infrastructure of the final solution. I will cover all the aspects required to visualize the entire system and highlight main points.

5.1 Functional Requirements

The table 5.1 captures all functional requirements that should be taken into account while developing the system.

Id	Description	Priority
FR1	The system must initialize and configure all sensors described in the corresponding configuration file	High
FR2	The system must initialize and configure all actuators described in the corresponding configuration file	High
FR3	The system must read data from sensors in given time intervals	High
FR4	The system must send collected sensor data from sensor controlling node to computational component	High
FR5	The system must store collected sensor data in a local database to keep data intact even in case of internet connection unavailability	High
FR6	The system must store collected sensor data in a cloud data storage for data analytics	Medium
FR7	The system must evaluate the collected sensor data and produce the outcome for actuator or actuators according to the defined rules	High
FR8	The system must send actuator commands to the actuator controlling node	High
FR9	The system must adjust actuators when the command is produced	High
FR10	The system must behave according to the rules defined in the corresponding file	Low

Table 5.1: Functional Requirements

5.2 Non-functional Requirements

- The system shall be able to operate without internet connection.
- The system shall be able to scale and accommodate new sensors or actuators.
- The system shall be able to recalibrate when sensor or actuator rules are modified even during runtime.
- The system shall be available non-stop.
- The system shall be consistent and maintain data integrity.
- The system shall be robust, highly performant and make calculations rather quickly.

5.3 Hydroponic System Environment Structure

In this section, I will go through the hydroponic system and describe how it functions. Figure 5.1 illustrates the entire hydroponic system structure, its individual components and the room where all components are located.

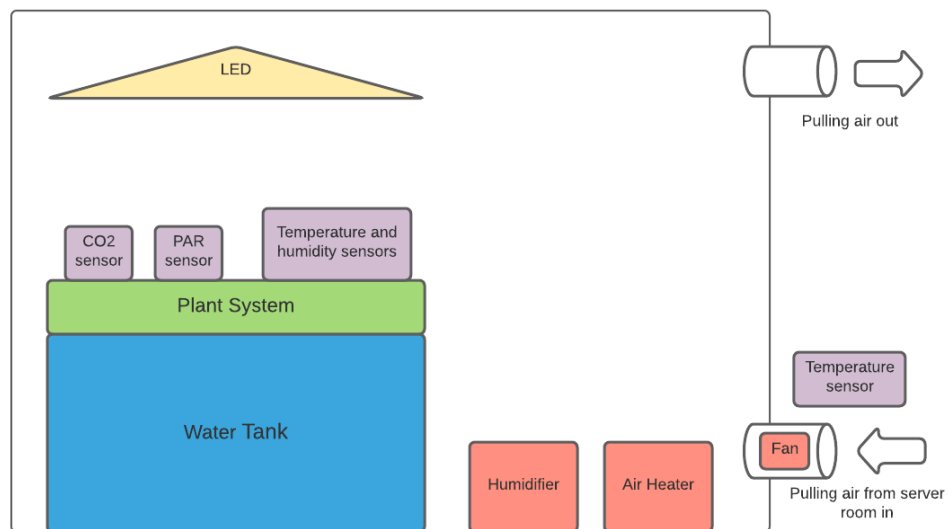


Figure 5.1: System Environment Schema

It can be seen that the hydroponic room is isolated and almost all the components required to control the system are located inside. The plant system is placed above the water tank, which contains nutrient-rich solution that is transported to the plant roots regularly.

The CO₂ and PAR sensors are located as close as possible to the plant system, so that the measurements are as accurate as possible. Moreover, PAR sensor

is placed wisely under the LED system to measure the correct values with respect to the light that plants actually receive. It was decided to use three air temperature sensors inside the room that are distributed evenly in the room. The three measured values are then processed by the corresponding service on Raspberry Pi. Due to the fact that humidity sensors are combined with air temperature sensors in one device, one of the room air temperature sensors is responsible for humidity measurements.

When it comes to actuators, the LED complex is placed on the ceiling of the room above the plants, thus distributing maximum amount of light evenly among all plants. Humidifier is responsible for increasing humidity values inside the room when it is needed, while air heater is in charge of raising air temperature in the room.

Although air heater is shown on the schema, it will not always be needed to regulate air temperature. Due to the fact that the room is located on the second floor above the server room where air temperature is usually relatively high, it was decided to use that air by pulling it in the room via tube. The tube has a fan that will control the amount of passing air from the server room. Moreover, the temperature of the air from the server room is also measured by one temperature sensor. The power of the fan is controlled by the system with respect to the air temperature values collected by three room temperature sensors and one tube temperature sensor. Finally, the room also has another tube, via which the air is pulled out of the room. Thus, air around the plant system is regularly circulated and renewed.

5.4 System Component Diagram

The UML component diagram illustrated in figure 5.2 describes individual system components, how they are connected and function altogether.

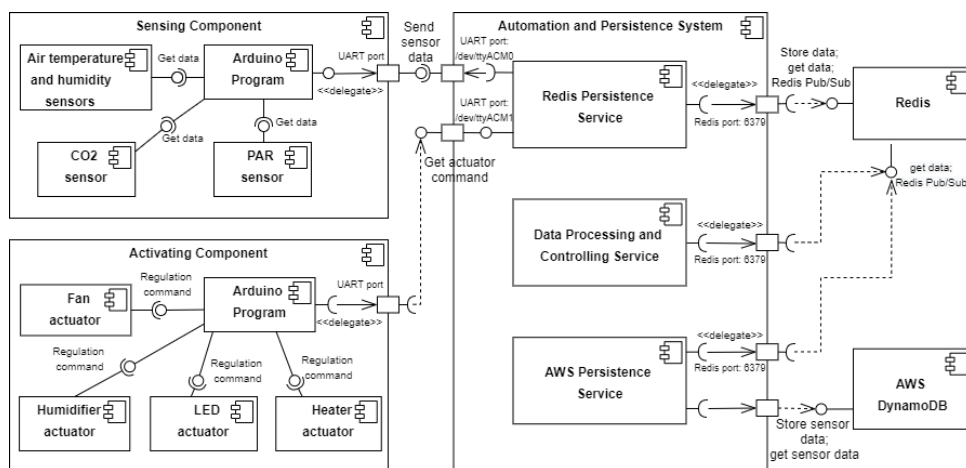


Figure 5.2: System UML Component Diagram

The system consists of three major components:

- *Sensing component.* It is responsible for working with sensors, their configurations and measurements of environmental parameters. The main component is Arduino program, which has most of the logic and collects data from sensors.
- *Activating component.* This is the part where the physical regulation of environmental parameters takes place. Arduino program component configures and sends signals to actuators to align the environmental parameters according to commands received from Automation and Persistence System or otherwise, Raspberry Pi.
- *Automation and Persistence System.* This is the brain of the entire hydroponic system. It consists of three components: Redis Persistence Service, Data Processing and Controlling Service and AWS Persistence Service. Each of these services are described in detail in the chapter Implementation.

5.5 System Deployment Diagram

The UML deployment diagram given in figure 5.3 describes how the system components will be physically deployed on the hardware.

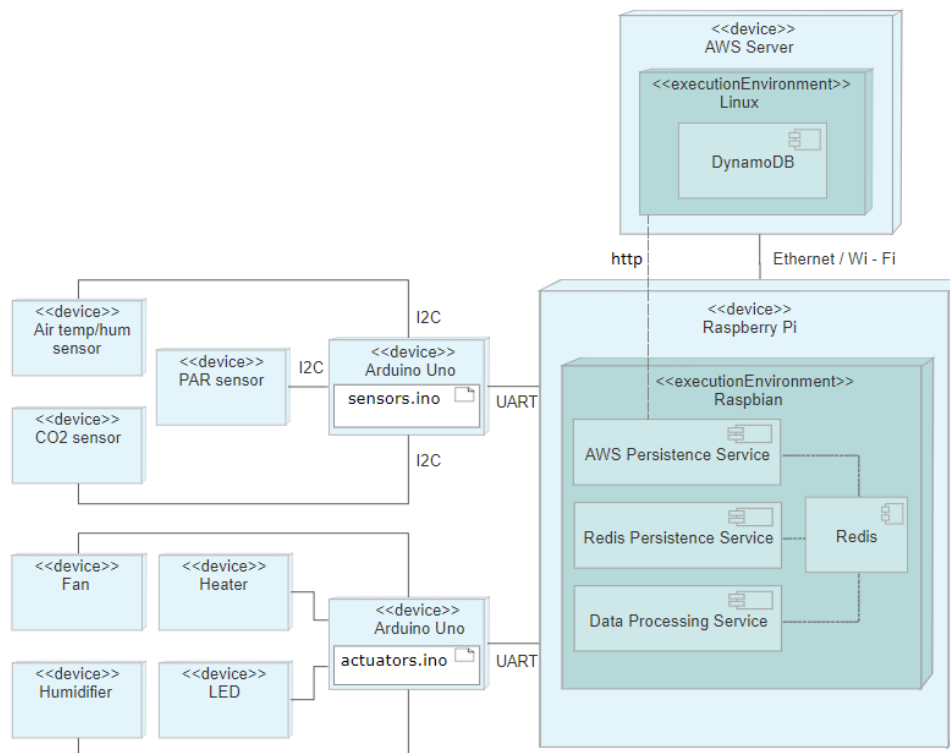


Figure 5.3: System UML Deployment Diagram

The hardware parts are classified into 5 types:

- *Sensors.* As it was already noted, the system consists of four types of sensors: air temperature and humidity sensors, CO2 sensor and PAR sensor. Each sensor is connected via I²C to the Arduino Uno microcontroller, which is responsible for handling sensors. In the actual system there are four air temperature and humidity sensors.
- *Actuators.* The following components comprise the actuator part of the system: fan, humidifier, led and air heater.
- *Microcontrollers.* Due to the fact that Arduino Uno has limited memory around 2 KB, it was decided to use two Arduino Uno boards. First one handles sensors, while second Arduino is responsible for activating actuators according to the commands received from Raspberry Pi. Not only this makes the logic clearer and increases cohesion of individual boards, but also makes the entire system faster by decreasing the reaction time of the microcontroller to commands.
- *Computational computer.* The brain of the system is Raspberry Pi. It makes most of the calculations on the collected data and persists data to local Redis database as well as AWS DynamoDB.
- *Databases.* The system uses two databases: Redis and AWS DynamoDB. The Redis is located on the Raspberry Pi and stores sensor data locally, while DynamoDB stores data in the cloud for data analytics. Furthermore, Redis is used as a message broker to implement the communication between services on Raspberry Pi.

■ 5.6 Communication Between Services

The main software part of the system located on Raspberry Pi is built from three services: Redis Persistence Service, AWS Persistence Service and Data Processing and Controlling Service. Since the system is being developed using microservices architecture, the individual services must somehow communicate. The communication mechanism is implemented using Redis as a message broker because it also implements publish-subscribe pattern.

■ 5.6.1 Publish-Subscribe Pattern

The publish-subscribe pattern is a way of passing messages to an arbitrary number of receivers. The senders of these messages, also known as *publishers*, do not explicitly identify the targeted recipients. Instead, the messages are sent out on a channel, on which any number of recipients, also called *subscribers*, can be waiting for them. [27]

5.6.2 Redis Pub/Sub Channels

The figure 5.4 illustrates the way services exchange data via channels using Redis Pub/Sub mechanism.

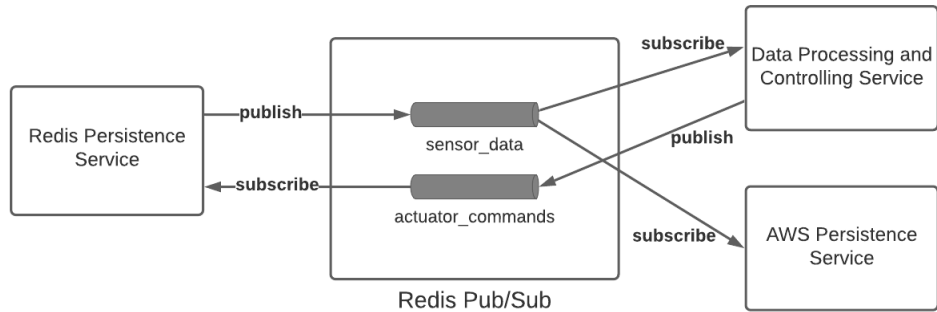


Figure 5.4: Redis Pub/Sub Channels

The channels are divided into two categories according to the data that pass through them. The first channel **sensor_data** is responsible for handling and distributing sensor data to subscribers. Redis Persistence service publishes sensor data received from Arduino Uno, while Data Processing and Controlling service and AWS Persistence service receive the data in real-time, as they are subscribed to the same channel. The second one **actuator_commands** transports data about actuator commands produced by Data Processing and Controlling service and sends them to Redis Persistence service. Subsequently, Redis Persistence service sends information to the corresponding Arduino Uno that manages actuators.

Chapter 6

Implementation

The overall automating system consists of several parts, mainly: programs on Arduino Uno, three services on Raspberry Pi and two databases. Each of these components are segregated and have certain responsibilities. In this chapter, I will describe the overall system implementation including the programs on two Arduinos, the services on Raspberry Pi and how databases are related to these services.

6.1 Arduino Uno Sketches

As I have already noted, the system is built from two Arduino Uno boards; first one responsible for working with sensors, whereas second Arduino handles actuators. In this section, I will go through the programs, also known as *sketches*, that are running on these two Arduino Uno boards and describe how they operate with hardware components, i.e., sensors and actuators, and communicate with the delegated service running on Raspberry Pi.

6.1.1 Sensors Sketch

First Arduino Uno runs a sketch, which is responsible for reading environmental data from sensors and sending them to Raspberry Pi. When the Arduino program starts, it requests configuration data for sensors including sensor name, type, address and time interval from the Redis Persistence service operating on Raspberry Pi. Subsequently, the service reads data from a configuration file and returns sensor data to Arduino Uno. After that, Arduino Uno has all information required to handle sensors.

First step is to configure all sensors. This means that each sensor is initialized on some given address using the corresponding library, in case if it is needed. After that, Arduino Uno constantly checks if any sensor measurements are required. In case if some measurement is needed, it reads data from the sensor and sends it to Redis Persistence Service. The UML sequence diagram illustrated in figure 6.1 describes the behaviour of this part of the system.

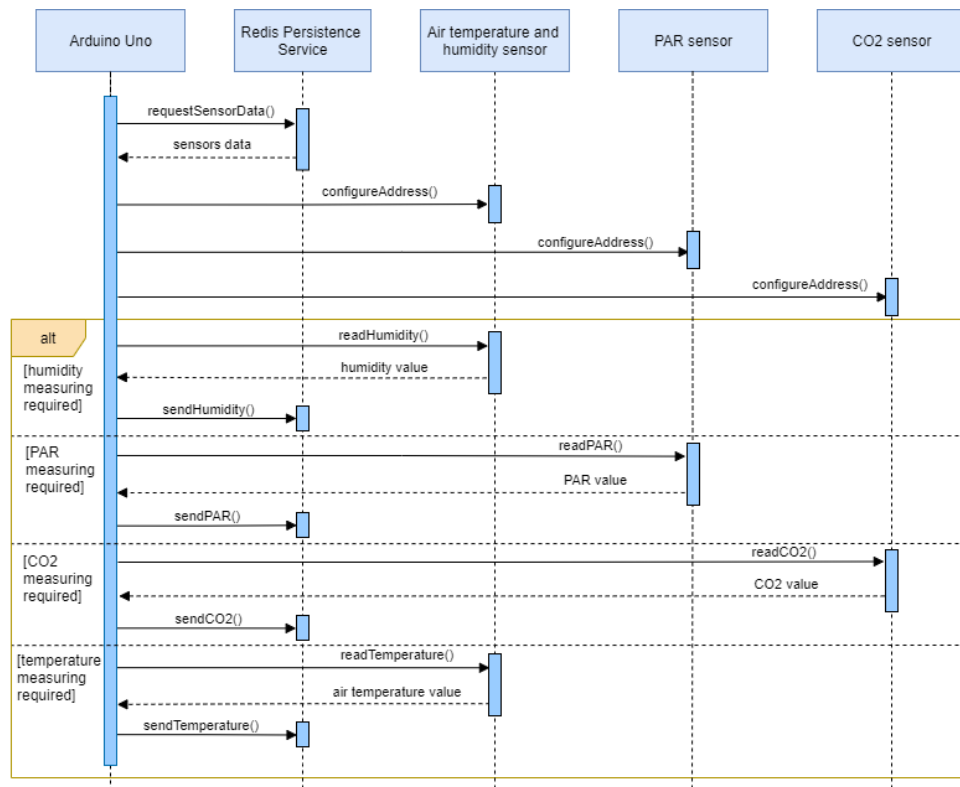


Figure 6.1: Sensor Handling UML Sequence Diagram

■ Setup Function

The following code snippet given in listing 6.1 shows what happens when Arduino Uno program is activated.

Listing 6.1: Setup Function

```

char config_data[500]; // used to fill in devices_data[] []

void setup() {
    Serial.begin(9600);
    Wire.begin();
    fetch_config_data();
    split(config_data);
    setup_devices();
}
  
```

When Arduino Uno board is turned on, the first thing that happens is calling `setup()` function. It retrieves sensors data from Raspberry Pi and transforms them into two-dimensional array `devices_data` using `fetch_config_data()` and `split()` helper functions. Each first-level entry of the array represents one sensor, while each second-level entry contains information about sensor name, type, address and time interval for the corresponding sensor. Finally, `setup_devices()` function initializes all sensors on the given addresses.

■ Loop Function

The code sample given below on listing 6.2 describes the functionality of measuring certain sensor data.

Listing 6.2: Loop Function

```
void loop() {
  for (int i = 0; i < number_of_sensors; i++) {
    const char* sensor_name = devices_data[i][0];
    const char* sensor_type = devices_data[i][1];
    unsigned long sensor_tempo = to_int(devices_data[i][3]);
    float sensor_value = 0;
    bool measuring_required = time_passed(last_sensor_timestamps[i],
      sensor_tempo);

    if (!measuring_required) continue;

    if (!strcmp(sensor_type, AIR_TEMP_SENSOR)) {
      if (!strcmp(sensor_name, SHT_SENSOR_NAME)) {
        sensor_value = sht20_lib.temperature();
      } else if (!strcmp(sensor_name, DHT_SENSOR_NAME_1)) {
        sensor_value = dht1.readTemperature();
      } else if (!strcmp(sensor_name, DHT_SENSOR_NAME_2)) {
        sensor_value = dht2.readTemperature();
      } else if (!strcmp(sensor_name, DHT_SENSOR_NAME_3)) {
        sensor_value = dht3.readTemperature();
      }
    } else if (!strcmp(sensor_type, HUMIDITY_SENSOR)) {
      sensor_value = sht20_lib.humidity();
    } else if (!strcmp(sensor_type, PAR_SENSOR)) {
      sensor_value = par_lib.measurePAR();
    } else if (!strcmp(sensor_type, CO2_SENSOR)) {
      sensor_value = co2_lib.readCO2PWM();
    }

    last_sensor_timestamps[i] = millis();
    send_data(sensor_name, sensor_type, sensor_value);
  }
}
```

The `loop()` function is called repeatedly on Arduino Uno board. It does the main job, which is collecting data from sensors. Each function call, we iterate over the `devices_data[] []` array and in each iteration representing current sensor, retrieve `sensor_name`, `sensor_type` and `sensor_tempo`, or else time interval, in which sensor data should be measured.

Next, the function checks whether the time interval of the current sensor is smaller than the difference between current time and last measured time for the iterating sensor. This is done using `last_sensor_timestamps` array that holds last timestamp for each sensor. Indices of the `last_sensor_timestamps` and `devices_data` correspond to the same sensor. Thus in each iteration, we can check whether it is time to make new sensor measurement. If this is

the case, then using the conditionals, function identifies what sensor type is in the current iteration and makes measurement. Finally, the last timestamp is updated and the measured value is send via serial link to Raspberry Pi, where it is accepted by Redis Persistence service.

6.1.2 Actuators Sketch

When it comes to the second Arduino Uno, its main goal is to listen to commands from Raspberry Pi and regulate actuators according to those commands. The principle is quite similar to the program described in the previous section, however, this program has to wait until some event triggers it. The event obviously is the actuator command. In the beginning of the program, Arduino requests configuration data for actuators including actuator name, type and address from Redis Persistence service. After that, the service fetches data from a configuration file and returns actuators data to Arduino Uno.

First step, after getting all the information about actuators, is to configure actuator addresses. Once this is done, the program is ready to listen to actuator commands and set actuators. The program constantly checks whether there is a message in a serial buffer. The UML sequence diagram given in figure 6.2 illustrates the behavior of physical actuator handling part of the system.

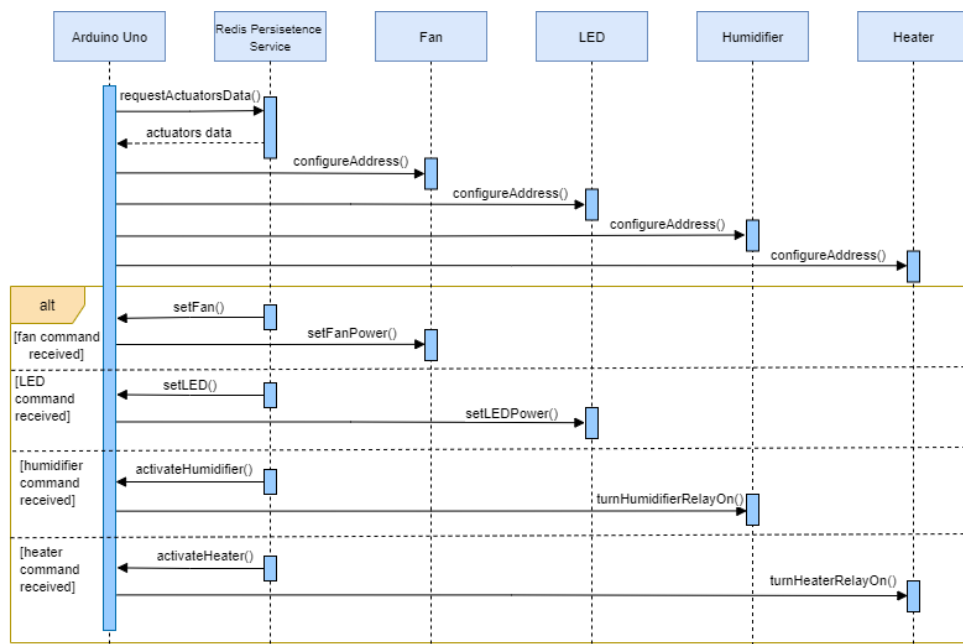


Figure 6.2: Actuators Handling UML Sequence Diagram

■ Setup Function

The following code illustrated in listing 6.3 reveals the initial setup of the program.

Listing 6.3: Setup Function

```
void setup() {
  Serial.begin(9600);
  Wire.begin();
  fetch_config_data();
  split(config_data);
  setup_devices();
}
```

As it can be seen, this part of the program is identical to the one, that runs on sensors handling Arduino. The only difference is in the messages they send to request data about devices. According to the following code snippet in listing 6.4, it is clear that the actuators handling Arduino sends HELLO_FROM_ACTUATORS string message, whereas sensor handling Arduino described in the previous section sends HELLO_FROM_SENSORS string.

Listing 6.4: Requesting Actuators Data

```
void fetch_config_data() {
  Serial.println("HELLO_FROM_ACTUATORS");

  while (true) {
    if (Serial.available() > 0) {
      String devices_data = Serial.readString();
      Serial.println(devices_data);
      int size = devices_data.length() + 1;
      devices_data.toCharArray(config_data, size);
      break;
    }
  }
}
```

After the message is sent, the program waits until Raspberry Pi sends reply containing data about actuators. This part is almost the same in the program that works with sensors. The only difference is in the message it sends to request data about sensors.

■ Loop Function

The following code snippet in listing 6.5 shows the way Arduino Uno constantly waits for the actuator commands by listening to the serial link. The Arduino `loop()` function calls `check_for_actuator_command()` function to find out whether there is data available in the serial buffer. When data arrive, it retrieves actuator type and value from the message. Next, the `process_command()` function is called to handle the actuator command.

Listing 6.5: Listening to Actuator Commands

```

void loop() {
    check_for_actuator_command();
}
void check_for_actuator_command() {
    while (Serial.available()) {
        String act_type = Serial.readStringUntil('\n');
        String act_command_value = Serial.readStringUntil('\n');
        process_command(act_type, act_command_value);
    }
}

```

The below code in listing 6.6 illustrates how the program handles the received actuator commands.

Listing 6.6: Processing Actuator Command

```

void process_command(String act_type, String command_value) {
    for (int i = 0; i < POSSIBLE_NUMBER_OF_DEVICES; i++) {
        const char* act_type = devices_data[i][1];
        int device_address = to_int(devices_data[i][2]);

        if (strcmp(received_act_type.c_str(), act_type)) continue;

        if (!strcmp(act_type, LED_ACT)) {
            analogWrite(device_address, act_command_value.toInt());
            return;
        }

        if (!strcmp(act_type, FAN_ACT)) {
            fanDimmer.setPower(act_command_value.toInt());
            return;
        }

        if (act_command_value.equals("0")) {
            digitalWrite(device_address, LOW); // turn relay off
            return;
        }

        digitalWrite(device_address, HIGH); // turn relay on
        return;
    }
}

```

The `process_command()` function receives two arguments: actuator type and command value. It goes through the `devices_data` array until it reaches the required actuator data and retrieves the address of that actuator. After that, it adjusts the required actuator by sending the corresponding signal. The possible signals are: `LOW` or `HIGH` for relay based actuators, percentage of power between 0% and 100% for fans based on TRIAC and number between 0 and 255 for LED, which is using pulse-width modulation method.

6.2 Redis Persistence Service

As it was already mentioned, Raspberry Pi runs 3 services: Redis Persistence service, AWS Persistence service and Data Processing and Controlling service. Each has a bit different responsibility. In this section, I will describe Redis Persistence service, which has an obligation to control the data flow between two Arduinos and the remaining services.

Redis Persistence service is an entry point to the business logic of the entire system. When the service starts, it establishes connection with Redis and subscribes to `actuator_commands` channel. Next, the service initializes Arduinos and waits for the request to share the data about sensors and actuators. These data are stored in separate configuration files, which are read when requests arrive.

When it comes to sensor data, the service triggers the message containing the measured environmental parameter value and information about the sensor. These data are then stored in local Redis database and subsequently published to `sensor_data` Redis Pub/Sub channel. The channel transports sensor data to AWS Persistence service and Data Processing and Controlling service, which then process these data.

When actuator command shows up from `actuator_commands` channel, the service sends the command further to actuators handling Arduino Uno. The following UML sequence diagram in figure 6.3 visualizes the described flow.

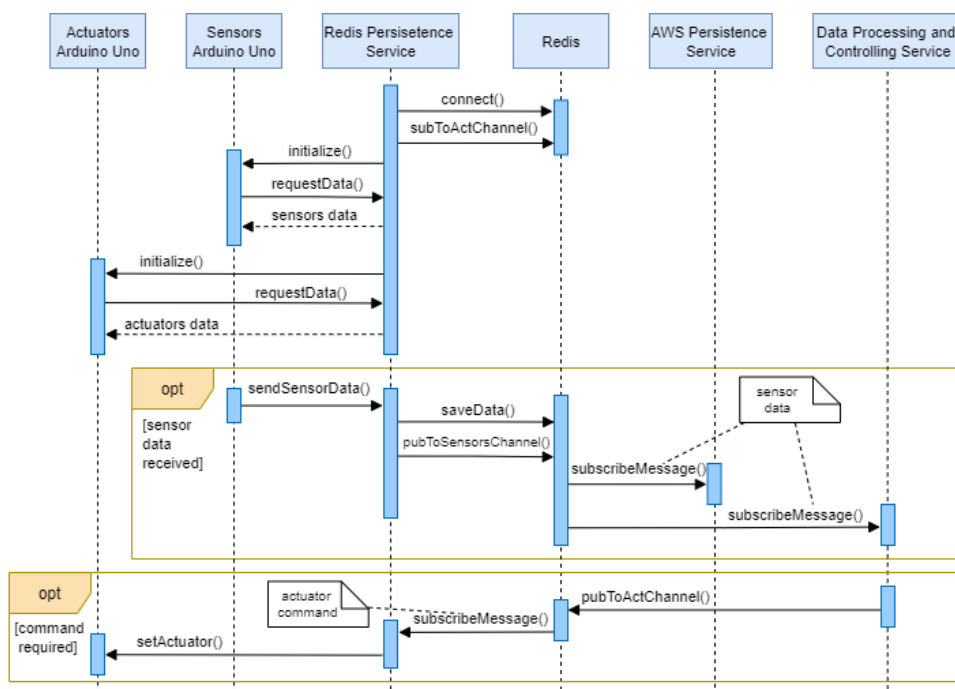


Figure 6.3: Redis Persistence Service UML Sequence Diagram

6.2.3 Persisting Sensor Data

Once sensor data arrives, Redis Persistence service processes these data and stores it in Redis. For this, there is a dedicated singleton class given in listing 6.9 that handles Redis related logic.

Listing 6.9: Redis Handling Class

```
class RedisHandler:
    instance = None

    @staticmethod
    def get_instance():
        """ Static access method for Singleton """
        if RedisHandler.instance is None:
            RedisHandler.instance = RedisHandler()
        return RedisHandler.instance

    def __init__(self):
        """ Virtually private constructor. """
        if RedisHandler.instance is not None:
            raise Exception("This class is a singleton!")

        self.redis_client = redis.StrictRedis(host='localhost',
            decode_responses=True, port=6379, db=0)
        self.redis_sub = self.redis_client.psubsub()
        self.redis_sub.subscribe(constants.PUB_SUB_ACT_CHANNEL_NAME)
```

The constructor initializes connection with Redis, enables publish-subscribe mechanism and subscribes to actuator_commands Pub/Sub channel. Once sensor data arrives, they get persisted by RedisHandler `save_data()` function. The code snippet is given in the following listing 6.10.

Listing 6.10: Persisting Sensor Data

```
def save_data(self, sensor_data):
    timestamp = sensor_data["timestamp"]
    sensor_name = sensor_data["sensor_name"]
    sensor_values = f'{sensor_name}_timestamps'
    new_timestamp_id = self.generate_id(timestamp)
    self.redis_client.hset(timestamp, mapping=sensor_data)
    self.redis_client.rpush(TIMESTAMPS_LIST_NAME, timestamp)
    self.redis_client.rpush(sensor_values, timestamp)
    self.redis_client.zadd(TIMESTAMPS_FOR_IDS_SORTED_SET_NAME,
        {timestamp: new_timestamp_id})
    self.redis_client.publish(PUB_SUB_SENSORS_CHANNEL_NAME,
        f'{timestamp} {str(new_timestamp_id)}')
```

Firstly, data are extracted and stored in Redis data structures mainly hash, list and sorted set. Each of them is needed to enable correct data sharing between services and to elevate data querying options. Finally, the information is sent to `sensor_data` channel and distributed to other services.

6.3 AWS Persistence Service

One of the major system requirements is that data collected by individual sensors are stored in a cloud database. This provides a great opportunity to introduce data analysis or even artificial intelligence. In this section, I will cover AWS Persistence service, which is responsible for storing sensor data in Amazon DynamoDB.

The Initial step AWS Persistence service takes is configuring connections with DynamoDB and Redis databases. It also subscribes to `sensor_data` Redis Pub/Sub channel to retrieve new messages about collected sensor data. When sensor data arrive, the service persists them to DynamoDB. As it was mentioned before, the system should function and maintain data integrity. This means that, even if internet connection is unavailable, the data that are being measured while there is no internet should not be lost. Thus, there should be a mechanism that will control the flow of the data and identify which data was and was not persisted to DynamoDB. The following figure 6.4 shows UML sequence diagram describing the flow of operations.

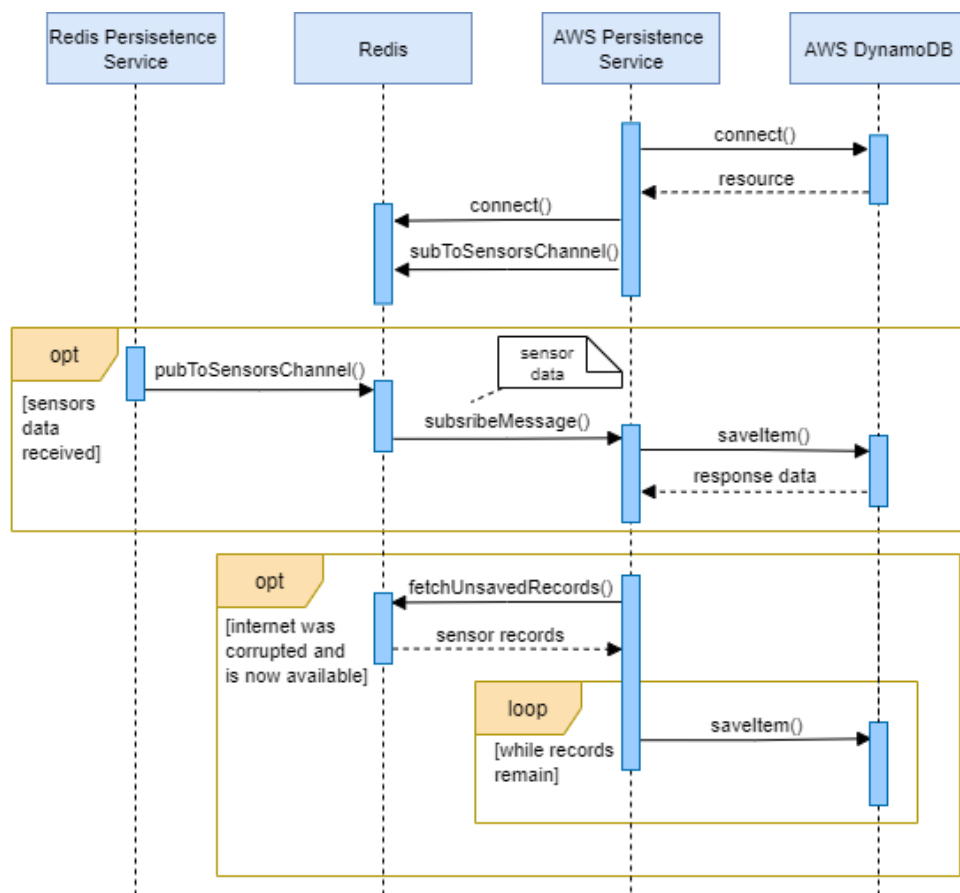


Figure 6.4: AWS Persistence Service UML Sequence Diagram

In case internet connection was lost and is available at the current moment, the service fetches sensor data from Redis that was not yet persisted to AWS DynamoDB. After that, the data are saved and the cloud data integrity is restored.

■ 6.3.1 Persisting Sensor Data

Once sensor data arrive, AWS Persistence services triggers and stores data in the cloud. There is a dedicated function illustrated in listing 6.11 that handles this job.

Listing 6.11: Persisting Sensor Data

```
def save_item(item):
    """
    Persists item to DynamoDB.
    If HTTP status is not 2xx or internet is corrupted returns False.
    :param item: item to persist
    :return: if error => False; else True
    """
    try:
        response = table.put_item(Item=item)
        resp_status = response[RESPONSE_METADATA][HTTP_STATUS_CODE]
        return str(resp_status).startswith("2")
    except botocore.exceptions.EndpointConnectionError:
        return False
```

The function calls `put_item` function from `boto3` module to persist data to DynamoDB. One of the most important parts of the AWS Persistence service is located in this code snippet. It checks whether data were stored successfully or not. In case data were not stored, it is detected in other parts of the service and handled.

■ 6.3.2 Managing Data Corruption

In this section, I will describe how the system behaves when problems with internet occur. The following code in listing 6.12 shows the way sensor data is retrieved from the channel and stored using `save_item()` function.

Listing 6.12: Retrieving Sensor Data From Channel

```
message = redis_handler.redis_sub.get_message()

if not message or message['type'] != "message":
    continue

timestamp, current_id = utils.split_timestamp_and_id(message)
sensor_data = redis_handler.redis_client.hgetall(timestamp)
is_saved = save_item(sensor_data)
```

However, connection to cloud may sometimes be inconsistent or even interrupted by failure in network. The following code snippet contains the logic that tackles this issue and restores sensor data integrity in DynamoDB.

Listing 6.13: Internet Connection Handling

```
if not is_saved and not disconnected:
    disconnected = True
    first_unsuccessful_id = current_id
elif is_saved and disconnected:
    disconnected = False
    last_unsuccessful_id = current_id - 1
    unsent_timestamps = redis_handler.get_data_in_range(
        min=first_unsuccessful_id, max=last_unsuccessful_id)
    save_all(unsent_timestamps)
```

This algorithm ensures that, when data are not persisted successfully, the id of this record will be registered and in a moment of network recovery, it will fetch all records from Redis ranging from `first_unsuccessful_id` to `last_unsuccessful_id`. Subsequently, these records are persisted to DynamoDB in `save_all()` function.

6.4 Data Processing and Controlling Service

Finally, in this section, I will describe the last, probably the most important service, which is Data Processing and Controlling service. This service is the brain of the entire hydroponic automated system. It is responsible for computing commands required for actuator regulation according to collected sensor data and predefined rules, in some cases, using mathematical functions to elect the most suitable value.

When the service starts, it establishes connection with Redis and subscribes to `sensor_data` Redis Pub/Sub channel. This channel delivers information about sensor measurements published by Redis Persistence service.

In the moment when data from sensor arrive, the service starts computing the corresponding actuator command value with respect to the environmental parameter value extracted from sensor data and rules related to this actuator type. When the computation finishes, the output command is published to `actuator_commands` Redis Pub/Sub channel along with the actuator type. After that, Redis Persistence service receives the subscribe message containing the computed actuator command and instructs Arduino Uno to manage the actuator. The following UML sequence diagram illustrated in figure 6.5 shows how the service interacts with peripheral entities and makes calculations on collected data.

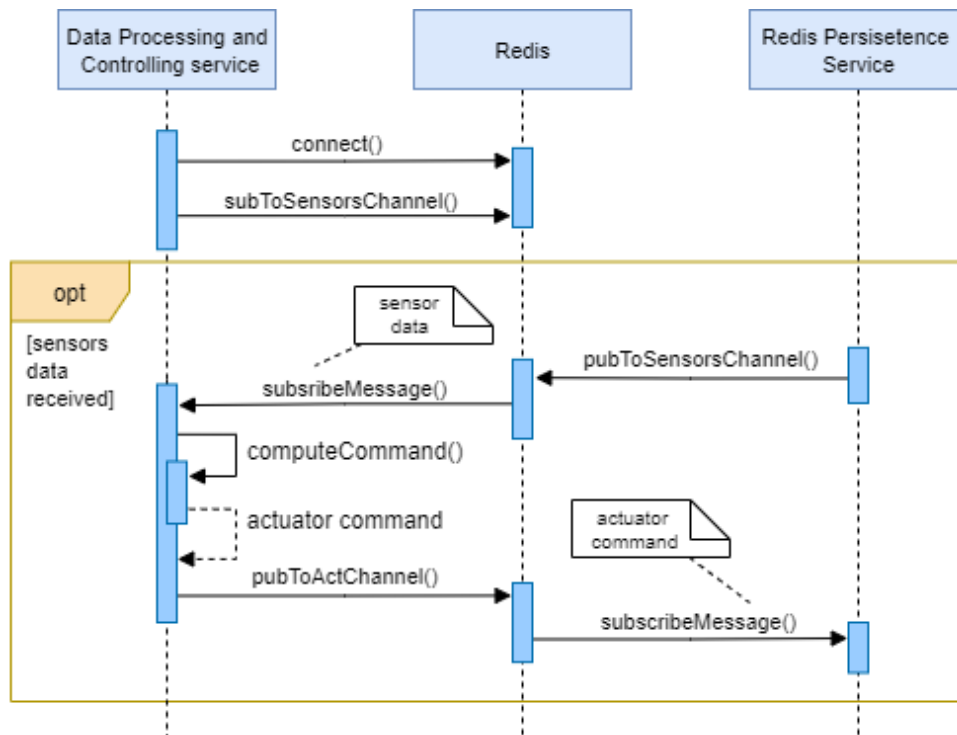


Figure 6.5: Data Processing and Controlling Service UML Sequence Diagram

6.4.1 Regulation Rules and Coefficients

To enable effective and sustainable regulation of environmental parameters, the service relies on a set of defined rules and coefficients. Each type of sensor data is handled by a corresponding actuator handler, which computes the output command according to the collected environmental parameter value. Moreover, during the computation, the service incorporates corresponding actuator rules and coefficients defined in separate JSON files.

The following listing 6.14 describes the coefficients used when calculating output actuator commands.

Listing 6.14: Actuator Coefficients

```

{
  "roomVolume": 6,
  "ledCoefficient": 0.5,
  "temperatureLossCoefficient": 2,
  "fanCoefficient": 0.6
}
  
```

The first coefficient `roomVolume` represents the volume of the room, where the hydroponic system is located. The `fanCoefficient` and `ledCoefficient` account for the performance and efficiency of fan and LED system respectively, while `temperatureLossCoefficient` represents the loss of air temperature

that is being transported from server room via tube. It is important to include these coefficients in some actuator-related calculations, which I will describe in the following section, to produce the most effective command value.

When it comes to actuator rules, the corresponding JSON file content is decomposed into three logical units: `led`, `airTemperature` and `humidity`. According to the listing 6.15, each property contains the rules related to the actuator type it represents.

Listing 6.15: Actuator Rules

```
{
  "led": {
    "required": 600,
    "led_power_lower_bound": 70,
    "led_power_upper_bound": 100,
    "sensor_type": "par_sensor",
    "actuator_type": "led_actuator",
    "actuator_relay_type": "led_actuator_relay",
    "turn_on_time": "08:00",
    "turn_off_time": "20:00"
  },
  "airTemperature": {
    "lower_bound": 24,
    "upper_bound": 26,
    "sensor_type": "air_temperature_sensor",
    "room_sensors_names": ["sht20", "dht1", "dht2"],
    "tube_sensor_name": "dht3",
    "actuator_up_type": "air_heater_actuator",
    "actuator_down_type": "fan_actuator",
    "desired_fan_power": 65,
    "fan_power_lower_bound": 40,
    "fan_power_upper_bound": 80
  },
  "humidity": {
    "required": 50,
    "sensor_type": "humidity_sensor",
    "actuator_type": "humidity_actuator"
  }
}
```

Each rule has `lower_bound`, `upper_bound` or `required` properties representing the boundaries for the environmental parameter value. They are used to identify whether the parameter value is in the tolerated range or not. Next properties observed in `led` and `air temperature` rules account for the actuator power boundaries, between which the actuator values can fluctuate. The next property `sensor_type` indicates, which sensor is responsible for measuring the required parameter value. Last but not least, `actuator_type` specifies the actuator type that is responsible for adjusting the given parameter.

The service is configured so that rules and coefficients JSON files can be modified even when the entire hydroponic system is operating since this is

one of the non-functional requirements. The service detects the last time the files were changed and, in case of modification, updates the rules and coefficients used for actuator command computation.

6.4.2 Environmental Parameters Regulation

In this section, I will describe the logic of individual actuator regulation according to the collected sensor data. The regulated actuators are: fan, LED system, humidifier and heater.

Fan

As it was already specified, fan actuator is responsible for transporting warm air from server room to the hydroponic room via specially installed tube. The fan itself is located inside the tube and controls the amount of air passing into the room. The power of the fan is adjusted using the output of the following formula

$$\Delta P = V_{room} \cdot c_{fan} \cdot (t_{req} - t_{room}) \cdot (t_{tube} - t_{req} - c_{loss}) \quad (6.1)$$

where V_{room} is the volume of the room, while c_{fan} represents the efficiency coefficient of the fan. The parameters t_{req} and t_{room} account for the desired air temperature and actual room air temperature respectively, whereas t_{tube} is the temperature of the air coming from the server room. t_{room} is calculated as the average of all collected room temperature values at a given moment. Finally, c_{loss} is a coefficient describing the temperature loss of the air that is being pulled in the room by the tube.

The output of this formula ΔP is added to the desired fan power specified in the rules file to produce the final command value. In case the final value for fan power is out of the defined boundaries, the value is aligned using the dedicated function. In our case, the desired fan power is set to 65% and adjusted every time the computation takes place. When the calculations are completed, the final value is sent to the corresponding fan actuator.

Heater

The heater regulation is closely related to the fan regulation using some parameters from its formula, however, managed in a bit different way. Compared to the fan, which is constantly operating and just switching power values according to the received commands, the heater is not always turned on. It is being controlled using a relay module responsible for turning the heater on and off. The following formula

$$t_{diff} = t_{tube} - t_{req} \quad (6.2)$$

outputs the value t_{diff} that is used to identify, whether the heater should be turned on or off.

In case $t_{diff} < c_{loss}$, where c_{loss} is a temperature loss coefficient described in the previous section, the heater is turned on. If $t_{diff} > c_{loss}$, the heater is switched off. Subsequently, the output command is sent to the heater actuator to stabilize the air temperature in the room.

■ LED

According to the fact that plants highly depend on the surrounding lights mainly Photosynthetically Active Radiation, the PAR values should be regulated accurately. For this, the LED power is adjusted using the output of the following formula

$$\Delta P = c_{led} \cdot (p_{req} - p_{room}) \quad (6.3)$$

where p_{req} and p_{room} are desired and actual PAR values respectively, whereas c_{led} is the efficiency coefficient of LED.

ΔP is then added to the current LED power to produce the final command value. If the value is out of the specified boundaries, it is adjusted using the error fixing function. The final value is indicated by P_{per} .

Due to the fact that LED is controlled with PWM signals that are on a scale of 0 – 255 and the P_{per} value produced from the previous calculations is expressed in percentages ranging from 0% to 100%, the value should be mapped to the [0, 255] interval using the following equation

$$P_{pwm} = \frac{P_{per} \cdot 255}{100} \quad (6.4)$$

After that, the output command value is sent to the LED actuator to regulate the lights.

■ Humidifier

The function of humidifier is quite similar to the heater. However, the moment when the humidifier should be turned on or off is detected in a different way. Just like the heater, the humidifier is controlled with the relay module, which is responsible for switching the power on and off.

According to the defined humidity rules, the humidity optimal value should fluctuate around the defined required value. In our case, the desired value is 50%. However, this can be easily modified in the rules due to your wish.

If the measured humidity value $h_{room} < h_{req}$ where h_{req} is the required humidity, the humidifier is turned on. Whereas, if $h_{room} > h_{req}$, the humidifier is turned off.

Chapter 7

System Evaluation

In this chapter, I will evaluate the achieved results and illustrate them using sensor data saved in AWS DynamoDB and the outputs of the logs for individual actuator command calculations.

7.1 Sensor Data Measurement and Persistence

According to one of the functional requirements the system should be able to persist data collected from sensors. Later, these data can be efficiently queried and utilized in analytics and frontend development. In this section, I will show the way sensor data is being persisted to Amazon DynamoDB and provide some actual samples.

The installed sensors collect the following environmental parameters:

- Humidity in %
- Air temperature in $^{\circ}C$
- Photosynthetically Active Radiation (PAR) in *nm*
- Carbon Dioxide (CO₂) in *ppm*

Every new record of sensor data related to the corresponding parameter is saved under the current timestamp; thus, these data can be used for time series analysis in the future.

The following figure 7.1 is a screenshot of a DynamoDB table that shows the way data collected by sensors is stored in AWS DynamoDB. We can see that each sensor value is stored with the specific timestamp, sensor name and type. Moreover, every new record of the same sensor type being inserted to the table will not delete the previous record. On the contrary, it will be saved as a new item, thus, preserving all previous collected data. These data then can be easily queried and used for statistics and other purposes.

system_id	timestamp	sensor_name	sensor_type	value
1	2021/05/17 16:23:33	sht_air_temp_sensor	air_temperature_sensor	24.75
1	2021/05/17 16:23:34	dht_sensor1	air_temperature_sensor	24.70
1	2021/05/17 16:23:35	dht_sensor2	air_temperature_sensor	24.80
1	2021/05/17 16:23:36	dht_sensor3	air_temperature_sensor	24.40
1	2021/05/17 16:23:53	humidity_sensor_name	humidity_sensor	45.68
1	2021/05/17 16:24:12	par_sensor_name	par_sensor	54.51
1	2021/05/17 16:24:26	co2_sensor_name	co2_sensor	934.00
1	2021/05/17 16:24:53	humidity_sensor_name	humidity_sensor	45.58
1	2021/05/17 16:24:54	sht_air_temp_sensor	air_temperature_sensor	24.86
1	2021/05/17 16:24:55	dht_sensor1	air_temperature_sensor	24.80
1	2021/05/17 16:24:56	dht_sensor2	air_temperature_sensor	24.80
1	2021/05/17 16:24:57	dht_sensor3	air_temperature_sensor	24.40
1	2021/05/17 16:25:23	par_sensor_name	par_sensor	54.42

Figure 7.1: AWS DynamoDB Sensor Data

Besides the fact that sensor data is being successfully persisted, we can also see that the sensor measurements happen regularly. As it was mentioned previously, measurements take place according to the time interval specified in the configuration file for sensors in Redis Persistence Service. For instance, air temperature value collected by `sht_air_temp_sensor` is measured every 80 seconds. This corresponds to the timestamps `16:23:33` and `16:24:54`. A similar pattern is observed in the rest of sensor measurements.

We can conclude that the data about environmental parameters are being successfully persisted and maintained. Moreover, sensors function correctly and measure parameters in the defined time intervals.

7.2 Parameter Regulation Outputs

In this section, I will evaluate the outputs produced by individual regulators described in Data Processing and Controlling Service chapter in different scenarios according to the collected data.

7.2.1 Air Temperature

Air temperature is being adjusted using the fan pulling the warm air from server room and the heater working together. One of the situations that can occur in the system is that the room air temperature is lower than the required temperature. Another one is that the room air temperature is higher than the desired temperature.

To make the evaluation clearer, I introduce the following variables:

- t_{req} - required air temperature, which in our case is 26.5 °C
- t_{room} - current room air temperature
- t_{tube} - server room air temperature being transported via tube
- ΔP - output value from fan power regulating formula

■ Low Room Air Temperature

It is obvious that if the room air temperature near plants is less than the required temperature, it should be somehow aligned. Due to the fact that, in most situations, air temperature in server room is quite high, the heater is not required for elevating the room air temperature. Instead, the fan will pull the warm air into the room and adjust the temperature. Nevertheless, I will go through every possible scenario and illustrate the output commands.

Scenario 1: $t_{room} < t_{req}$ and $t_{tube} > t_{req}$

The logs from Data Processing and Controlling service given in figure 7.2 show how air temperature values are collected from all three room air temperature sensors and one tube sensor. When all values are collected, the computation of actuator commands starts and calculates the output commands for fan and heater.

```

May 17 15:44:49 pi air_temperature_handler[4630]: [INFO] Air temperature handler activating
May 17 15:44:49 pi air_temperature_handler[4630]: [INFO] Received from sensor sht_air_temp_sensor value: 24.48
May 17 15:44:49 pi air_temperature_handler[4630]: [INFO] 2 room air temperature measurements left
May 17 15:44:50 pi air_temperature_handler[4630]: [INFO] Air temperature handler activating
May 17 15:44:50 pi air_temperature_handler[4630]: [INFO] Received from sensor dht_sensor1 value: 24.3
May 17 15:44:50 pi air_temperature_handler[4630]: [INFO] 1 room air temperature measurements left
May 17 15:44:51 pi air_temperature_handler[4630]: [INFO] Air temperature handler activating
May 17 15:44:51 pi air_temperature_handler[4630]: [INFO] Received from sensor dht_sensor2 value: 24.3
May 17 15:44:51 pi air_temperature_handler[4630]: [INFO] 0 room air temperature measurements left
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Air temperature handler activating
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Received from sensor dht_sensor3 value: 30.9
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] All air temperature sensor values collected
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Required air temperature: 26.5
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Room air temperatures mean value: 24.36
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Tube air temperature value: 30.9
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] produced FAN delta power from formula: 9
May 17 15:44:52 pi air_temperature_handler[4630]: [INFO] Sending produced FAN power to fan actuator with value: 79
May 17 15:44:52 pi redis_handler[4630]: [INFO] Sending command to fan_actuator with value 79
May 17 15:44:52 pi redis_handler[4630]: [INFO] Sending command to air_heater_actuator with value 0

```

Figure 7.2: Low Room and High Server Room Air Temperature Logs

t_{room} is equal to 24.36 °C, whereas t_{tube} is 30.9 °C, which is more than enough to adjust the room air temperature using fan. The fan power should increase, so that more warm server room air passes into the room. The fan actuator formula produced $\Delta P = 9$, which means that the fan should increase the power by 9%. On the other hand, we can also observe that the heater-related calculations instructed to turn the heater off, because it is redundant due to the fact that the server room air temperature is warm enough.

In this scenario the regulation of actuators was successful.

Scenario 2: $t_{room} < t_{req}$ and $t_{tube} < t_{req}$

In case server room air temperature is lower than the required air temperature, that air cannot be used to increase the room air temperature. The following figure 7.3 shows the logs for this scenario.

```

May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] Required air temperature: 26.5
May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] Room air temperatures mean value: 24.31
May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] Tube air temperature value: 24.3
May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] produced FAN delta power from formula: -16
May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] Sending produced FAN power to fan_actuator with value: 54
May 17 15:40:52 pi redis_handler[4630]: [INFO] Sending command to fan_actuator with value 54
May 17 15:40:52 pi air_temperature_handler[4630]: [INFO] Sending command to turn air_heater_actuator relay ON
May 17 15:40:52 pi redis_handler[4630]: [INFO] Sending command to air_heater_actuator with value 1

```

Figure 7.3: Low Room and Low Server Room Air Temperature Logs

t_{room} is equal to 24.31 °C, while t_{tube} is 24.3 °C, which is not enough to elevate the room air temperature. On the contrary, server room air is colder; thus, the fan power should decrease, so that less server room air can pass into the room. The fan actuator formula produced $\Delta P = -16$, which means that the fan power should be decreased by 16%. It can be seen that the computation instructs to turn the heater on to adjust room air temperature.

In this scenario the regulation of actuators was successful.

■ High Room Air Temperature

If the room air gets warmer, its temperature should be regulated and returned to the desired range. The following scenarios describe how the system behaves in this case.

Scenario 1: $t_{room} > t_{req}$ and $t_{tube} < t_{req}$

If the server room air temperature t_{tube} is lower than required, this air is used to adjust the room air temperature. The following figure 7.4 shows the logs for this scenario.

```

May 17 16:55:40 pi air_temperature_handler[4630]: [INFO] Required air temperature: 26.5
May 17 16:55:40 pi air_temperature_handler[4630]: [INFO] Room air temperatures mean value: 29.89
May 17 16:55:40 pi air_temperature_handler[4630]: [INFO] Tube air temperature value: 24.3
May 17 16:55:40 pi air_temperature_handler[4630]: [INFO] produced FAN delta power from formula: 25
May 17 16:55:40 pi air_temperature_handler[4630]: [INFO] Sending produced FAN power to fan_actuator with value: 4
May 17 16:55:40 pi redis_handler[4630]: [INFO] Sending command to fan_actuator with value 4

```

Figure 7.4: High Room and Low Server Room Air Temperature Logs

t_{room} is equal to 29.89 °C, while t_{tube} is 24.3 °C, which is just right to reduce the room air temperature t_{room} to the required t_{req} . The fan actuator formula outputted $\Delta P = 25$, which means that the fan power should be intensified by 25%. Thus, the fan will pull more cold server room air into the main room and align the air temperature.

In this scenario the regulation of actuators was successful.

Scenario 2: $t_{room} > t_{req}$ and $t_{tube} > t_{req}$

In case both t_{room} and t_{req} are higher than the required air temperature value, there is no need to pull server room air. Instead, there should a mechanism that will chill the room air temperature, which is not yet installed in the system but will definitely be present in the close future. However, we can still analyze how the system behaves in this situation shown in figure 7.5.

```
May 17 17:10:21 pi air_temperature_handler[4630]: [INFO] Required air temperature: 26.5
May 17 17:10:21 pi air_temperature_handler[4630]: [INFO] Room air temperatures mean value: 30.16
May 17 17:10:21 pi air_temperature_handler[4630]: [INFO] Tube air temperature value: 32.1
May 17 17:10:21 pi air_temperature_handler[4630]: [INFO] produced FAN delta power from formula: -23
```

Figure 7.5: High Room and High Server Room Air Temperature Logs

t_{room} is equal to 30.16 °C, whereas t_{tube} is 32.1 °C. The produced value from fan actuator formula is $\Delta P = -21$ meaning that the fan power should be lowered by 21%. This is acceptable since less warm server room air will pass into the room, however, not ideal to resolve the issue.

In this scenario the regulation of actuators was satisfactory but needs to be improved.

7.2.2 Lights

In this section, I will evaluate the regulation of PAR values. The lights around plants are regulated by a specialized LED system. The power of the LED is regulated every time the PAR sensor reads the value and approximated to the required value, which in our case is 600 *nm*. The following logs in figure 7.6 illustrate the regulation of LED power.

```
May 17 15:44:19 pi led_handler[4630]: [INFO] LED handler activating
May 17 15:44:19 pi led_handler[4630]: [INFO] Received from sensor par_sensor_name value: 54.51
May 17 15:44:19 pi led_handler[4630]: [INFO] Sending produced LED power signal to led_actuator with value: 100
May 17 15:44:19 pi redis_handler[4630]: [INFO] Sending command to led_actuator with value 255
```

Figure 7.6: PAR Regulation Logs

The collected PAR value is equal to 54.51 *nm*, which is extremely small amount of light for plants. Therefore, produced ΔP from the led actuator formula is much higher than 100%; thus, the power of the LED is set to 100%.

We can conclude that the LED regulation was successful.

7.2.3 Humidity

Since humidifier is managed in the similar way as is the heater, there is no need to examine complex computational formulas for producing delta power values. Instead, the service checks whether the collected humidity value is lower or higher than the required value. In our case, the required humidity value is set to 50%. The following scenarios describe how the system operates in these situations.

To make the evaluation clearer, I introduce the following variables:

- h_{req} - required humidity, which in our case is 50%
- h_{room} - current humidity in the room

Scenario 1: $h_{room} < h_{req}$

In case current humidity value h_{room} is less than the required h_{req} , the humidity in the room should increase; thus, the humidifier is turned on. The logs in figure 7.7 show how the system reacts in this scenario.

```
May 19 16:31:58 pi humidity_handler[6628]: [INFO] Humidity handler activating
May 19 16:31:58 pi humidity_handler[6628]: [INFO] Received from sensor humidity_sensor_name value: 49.88
May 19 16:31:58 pi humidity_handler[6628]: [INFO] Humidity required: 50
May 19 16:31:58 pi redis_handler[6628]: [INFO] Sending command to humidity_actuator with value 0
May 19 16:31:58 pi humidity_handler[6628]: [INFO] Sending command to turn humidity_actuator relay ON
```

Figure 7.7: Low Humidity Logs

The actual humidity value is equal to 49.88%, which is lower than the defined required humidity value 50%. Thus, the humidifier is turned on to increase relative humidity in the hydroponic room.

In this scenario the regulation of the humidifier was successful.

Scenario 2: $h_{room} > h_{req}$

If the actual humidity value h_{room} is greater than the required h_{req} , the humidifier is turned off. The logs for this scenario are given in figure 7.8.

```
May 19 16:33:58 pi humidity_handler[6628]: [INFO] Humidity handler activating
May 19 16:33:58 pi humidity_handler[6628]: [INFO] Received from sensor humidity_sensor_name value: 50.8
May 19 16:33:58 pi humidity_handler[6628]: [INFO] Humidity required: 50
May 19 16:33:58 pi redis_handler[6628]: [INFO] Sending command to humidity_actuator with value 1
May 19 16:33:58 pi humidity_handler[6628]: [INFO] Sending command to turn humidity_actuator relay OFF
```

Figure 7.8: High Humidity Logs

The current humidity value is 50.8%, which is higher than the desired value. According to the logs, the humidifier is switched off.

In this scenario the regulation of the humidifier was successful.



Chapter 8

Conclusion

The goal of this thesis was to construct a reliable automated system for hydroponically grown plants by implementing monitoring and regulation of environmental parameters.

I would like to say that this work has accomplished its goals. This paper includes the study of hydroponics in general, analysis of hardware, software and available technologies, design and implementation of automated hydroponic system and finally, the evaluation of the entire system operation. In the last chapter I have illustrated how the final product functions and given some outputs of the installed system logs. I can conclude that the solution has been successfully constructed, automated and integrated into the regular hydroponic system making it significantly more effective and efficient. The environmental parameters around the installed system are being adequately measured and regulated by the intelligently developed service.

For me personally, this work has given enormous knowledge in different spheres, with which I was not quite familiar. I have gained great experience while configuring hardware and incorporating it into the system. Moreover, developing the system based on microservices architecture gave me a better understanding of the way the services can communicate and operate side by side. Last but not least, I have expanded my knowledge of databases mainly while working with Redis and Amazon DynamoDB, which were used in the implementation of the system.

Nevertheless, there is a lot of room for further development and improvement, which would bring the system quality to another level. The regulation of parameters will become even more effective, if the Artificial Intelligence is introduced into the system. Also, there is a great opportunity to develop a front-end application that will be summarizing and displaying the measured data in the form of graphs and charts. Finally, the collected sensor values can be used to create time series data for data analysis to examine the environmental changes around the plants.

Appendix A

Source Code



To view the system source code visit the project git repository by scanning the above QR code. You can also click on it and redirect to the repository page.

Appendix B

Installed System Images



Figure B.1: Growing Plants

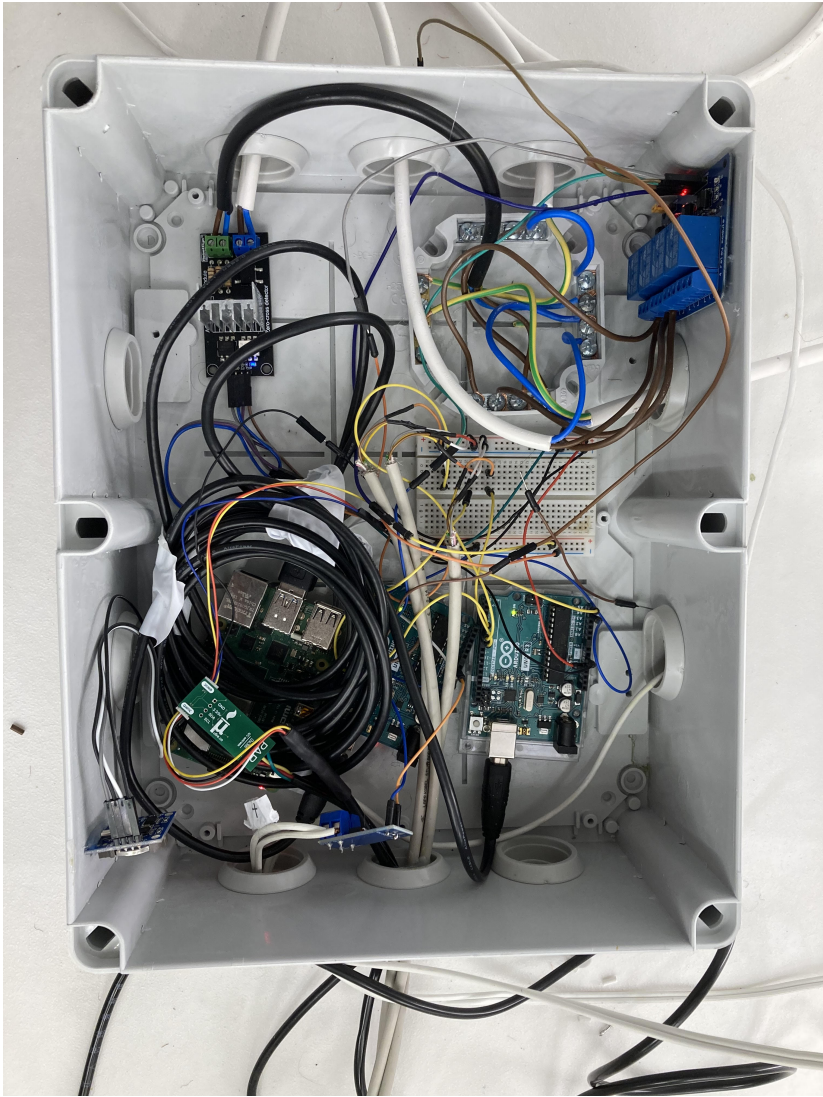


Figure B.2: Hardware Box



Appendix C

List of Abbreviations

- PAR - Photosynthetically Active Radiation
- NFT - Nutrient Film Technique
- PVC pipe - polyvinyl chloride pipe
- LECA - light expanded clay aggregate
- LED - Light Emitting Diode
- EC - Electrical Conductivity
- RPI - Raspberry Pi
- CPU - Central Processing Unit
- I/O - Input/output
- USB - Universal Serial Bus
- IoT - Internet of Things
- PWM - Pulse width modulation
- IDE - Integrated Development Environment
- SD - Secure Digital
- OS - Operating System
- I²C - Inter-Integrated Circuit
- UART - Universal Asynchronous Receiver and Transmitter
- SPI - Serial Peripheral Interface
- SQL - Structured Query Language
- API - Application Programming Interface
- UML - Unified Modeling Language
- AWS - Amazon Web Services
- AI - Artificial Intelligence

Appendix D

Bibliography

- [1] Nikita Bondarev. *Monitoring simulation and automated regulation of parameters of hydroponic system solution*. Bachelor Thesis. Czech Technical University in Prague, Faculty of Electrical Engineering and Technology, 2021.
- [2] J. Benton Jones Jr. *Hydroponics: A Practical Guide for the Soil-less Grower*. 2nd Edition. Boca Raton: CRC Press, 2016. ISBN 978-0849331671.
- [3] Textier William. *Hydroponics for Everybody: All About Home Horticulture*. San Francisco: Quick American Archives, 2015. ISBN 978-2845941205.
- [4] M. Max. *Advantages and Disadvantages of Hydroponics*. 2021. [online]. [cit. 2021-03-15]. <https://www.trees.com/gardening-and-landscaping/advantages-disadvantages-of-hydroponics>.
- [5] *6 Types of Hydroponic Systems*. 2019. [online]. [cit. 2021-03-20]. <https://sensorex.com/blog/2019/10/29/hydroponic-systems-explained/>.
- [6] Pak. J. Agri., Agril. Engg., Vet. Sc. *Hydroponics: Key to Sustain Agriculture in Water Stressed and Urban Environment*. 2006. [online]. [cit. 2021-03-20]. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.514.4323&rep=rep1&type=pdf>.
- [7] Keith Roberto. *How-To Hydroponics*. 4th Edition. Futuregarden, Inc. 2003. ISBN 978-0967202617.
- [8] *Hydroponics For Beginners - The Definitive Guide*. 2018. [online]. [cit. 2021-03-24]. <https://www.trees.com/gardening-and-landscaping/hydroponic-gardening>.
- [9] Howard M. Resh. *Hydroponic Food Production. A Definitive Guidebook for the Advanced Home Gardener and the Commercial Hydroponic Grower*. 7th Edition. Boca Raton: CRC Press, 2012. ISBN 978-1439878675.
- [10] Holland Horticulture. *Why EC Is important in hydroponics*. 2017. [online]. [cit. 2021-03-27]. <https://www.hydroponics.co.uk/news/why-ec-is-important-in-hydroponics/>.

- [11] C Eng Faruk Bin Poyen. *Raspberry Pi and its use in IoT applications*. 2019. [online]. [cit. 2021-04-05]. https://www.researchgate.net/publication/330200556_Raspberry_Pi_and_its_Use_in_IoT_Applications.
- [12] Yusuf Abdullahi Badamasi. *The Working Principle of an Arduino*. IEEE, 2014. [online]. [cit. 2021-04-06]. <https://ieeexplore.ieee.org/abstract/document/6997578>.
- [13] Ravi Teja. *What are the differences between Raspberry Pi and Arduino Uno?* 2021. [online]. [cit. 2021-04-09]. <https://www.electronicshub.org/raspberry-pi-vs-arduino/>.
- [14] André Glória, Francisco Cercas, Nuno Souto. *Comparison of communication protocols for low cost Internet of Things devices*. IEEE, 2017. [online]. [cit. 2021-04-11]. <https://ieeexplore.ieee.org/abstract/document/8088226>.
- [15] Robin Mitchell. *Common Communication Peripherals on the Arduino: UART, I2C, and SPI*. 2018. [online]. [cit. 2021-04-11]. <https://maker.pro/arduino/tutorial/common-communication-peripherals-on-the-arduino-uart-i2c-and-spi>.
- [16] D. Anastasia. *Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless?* 2019. [online]. [cit. 2021-04-15]. <https://rubygarage.org/blog/monolith-soa-microservices-serverless#:~:text=Monolithic%20apps%20consist%20of%20interdependent,and%20feature%20rapid%20continuous%20development>.
- [17] Mahesh Parahar. *Difference between monolithic and microservices architecture*. 2020. [online]. [cit. 2021-04-15]. <https://www.tutorialspoint.com/difference-between-monolithic-and-microservices-architecture#:~:text=Monolithic%20architecture%20is%20built%20as,is%20usually%20one%20code%20base.&text=Microservices%20architecture%20is%20built%20as%20small%20independent%20module%20based%20on%20business%20functionality>.
- [18] Sitalakshmi Venkatraman, Kiran Fahd, Samuel Kaspi, Ramanathan Venkatraman. *SQL Versus NoSQL Movement with Big Data Analytics*. 2016. [online]. [cit. 2021-04-19]. <http://j.mecs-press.net/ijitcs/ijitcs-v8-n12/IJITCS-V8-N12-7.pdf>.
- [19] Amazon Web Services. *What is DynamoDB?*. [online]. [cit. 2021-04-19]. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>.
- [20] MongoDB Documentation. *Sharding*. [online]. [cit. 2021-04-19]. <https://docs.mongodb.com/manual/sharding/>.
- [21] Redis Documentation. *Data types*. [online]. [cit. 2021-04-19]. <https://redis.io/topics/data-types>.

- [22] Michael McRoberts. *Beginning Arduino*. 2nd Edition. New York: Apress, 2011. ISBN 978-1430250166.
- [23] Manoel Carlos Ramon. *Arduino IDE and Wiring Language*. 2014. [online]. [cit. 2021-04-21]. https://link.springer.com/chapter/10.1007/978-1-4302-6838-3_3.
- [24] Arduino.cc. *Wire Library*. [online]. [cit. 2021-04-21]. <https://www.arduino.cc/en/reference/wire>.
- [25] Matt Richardson, Shawn Wallace. *Getting Started with Raspberry Pi*. 1st Edition. Sebastopol: O'Reilly Media, Inc., 2012. ISBN 978-1449344214.
- [26] *Serial Communication between Python and Arduino*. [online]. [cit. 2021-04-21]. <https://create.arduino.cc/projecthub/ansh2919/serial-communication-between-python-and-arduino-e7cce0>
- [27] Redis Documentation. *Redis Pub/Sub*. [online]. [cit. 2021-04-25]. <https://redis.io/topics/pubsub>.