Master thesis : Redesigning the automation and intelligence platform for indoor growing systems

Auteur : Pinchard, Bastien
Promoteur(s) : Boigelot, Bernard; Lebeau, Frédéric
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "intelligent systems"
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/14380

University of Liège
Faculty of Applied Sciences

Master in Computer Science with a Professional Focus on Intelligent Systems

Redesigning the automation and intelligence platform for indoor growing systems

Master thesis submitted in order to obtain the degree of master in Computer Science by

Bastien Pinchard

Montefiore Institute
Academic Supervisor:
Bernard Boigelot

Gembloux Agro-Bio Tech
Academic Supervisor:
Frédéric Lebeau

Academic year 2021-2022
June 9, 2022

Abstract

Agricultural techniques have always evolved by embracing the industry changes that took part according to the human technological evolution. Nowadays, Industry 4.0 represents the fourth industrial revolution leading to Agriculture 4.0. This last evolution is mainly defined by the fusion of many emerging technologies such as the Internet of things, advanced electronics and robotics, big data, and artificial intelligence. The new Agriculture 4.0 ecosystem is thus characterized by real-time farm management, a high degree of automation, and data-driven intelligent decision-making. The emerging smart plant factory concept is the real world realization of this new agricultural paradigm. This work proposes a new automation and intelligence platform that is defined as a basic hardware pattern and software structure upon which information and communication systems that serve as a base for various services are built. This platform has the objective of accommodating most functions needed by these state of the art farms. More precisely, the main target is to organize the communications and roles of each implemented module. To do so, a proof of concept of the monitoring hardware and a minimum viable product of the control software are implemented. Moreover, this project was tested with real world cultivation experiments thanks to a custom made cultivation tray. By having a multisectoral approach and by studying the modularity of the systems, the conducted development were shown to improve upon the currently available solutions regarding the technical documentation, edge and cloud computing compatibility, and hardware and software flexibility.

Acknowledgments

First and foremost, I would like to express my gratitude to my academic supervisors, Professors Bernard Boigelot and Frédéric Lebeau, for their supervision, their guidance throughout this master thesis, and for their participation to this personally defined research subject.

Then, I would like to thank again Professor Frédéric Lebeau for the sharing of its extensive knowledge of the modern agricultural fields and for the important advice given at the early stage of the thesis.

Next, I would also like to thank again Professor Bernard Boigelot for the invaluable feedback shared during the most crucial stage of the thesis and for the essential objective management provided at all time.

Furthermore, I extend my gratitude to the other academic employees who accepted interviews and meetings that have considerably improved this work overall quality.

Last but not least, I want to do a sincere special thank towards my family and friends who genuinely supported me and greatly helped me at various levels during the writing of this master thesis and the development of its intrinsic project.

Contents

Acronyms

1W One-Wire. 23, 30, 31
3D Three Dimensional. 7, 16

A4 Agriculture 4.0. 1–4, 12–14
ADC Analog to Digital Converter. 28, 30, 32, 44, 45, 67, 68
AI Artificial Intelligence. 1, 3, 13, 14
AIP Automation and Intelligence Platform. 3, 14–35, 53–58, 64, 67, 68

CEA Controlled Environment Agriculture. 2, 5–9, 11–13, 16, 18
CSM Cultivation System Module. 2, 18–20, 30, 34, 38, 39, 46–49, 51

DB DataBase. 13, 15, 24, 27

EC Electrical Conductivity. 10, 30, 31, 45, 65, 67
EMS Electronic Management System. 17, 20, 34–36, 39, 40, 42, 43, 45, 46, 55–58, 61, 62, 64

GPIO General Purpose Input/Output. 23, 28, 31, 32, 45, 68

HAT Hardware Attached on Top. 23, 24, 28–31, 45, 67
HS Hydroponics System. 5–9, 12–16, 18, 57

I/O Input/Output. 22, 23, 29
ICT Information and Communication Technology. 1, 3, 5, 11, 15, 17, 20, 28, 56, 58
IoT Internet of Things. 1, 3, 13, 14, 16
I²C Inter-Integrated Circuit. 23, 29–33, 42, 43, 45, 56, 68

MCU MicroController Unit. 22, 23
MD Management Dashboard. 17, 20, 25, 34, 35, 45–52, 54–58, 61, 62
ML Machine Learning. 13–15, 23, 25, 57
MVP Minimum Viable Product. 4, 17, 34, 54, 56–58

ORP Oxidation-Reduction Potential. 30, 31, 45, 67, 68

PF Plant Factory. 2, 5, 8–14, 16, 17, 21, 58
pH Potential of Hydrogen. 10, 14, 29–31, 44, 45, 64, 67, 68
POC Proof Of Concept. 4, 17, 28–31, 45, 54, 56–58, 67
PWM Pulse-Width Modulation. 31, 32

RPI Raspberry Pi. 23–25, 27, 28, 31–33, 45, 54, 56, 67

SBC Single Board Computer. 22, 23, 28, 57
SOTA State Of The Art. 4, 6, 11, 13, 15, 21
SPF Smart Plant Factory. 2–5, 11–18, 20, 21, 28, 30, 34, 36, 39, 45–48, 55–58

VF Vertical Farming. 5, 7–9, 12, 16, 18

Chapter 1

Introduction

Agricultural techniques have always evolved by embracing the industry changes that took part according to the human technological evolution. Firstly, these methods went from manual traditional farming techniques used up to the end of the 19th century (Agriculture 1.0) to the first machinery used in Agriculture 2.0 following the first industrial revolution (Industry 1.0). Then, the farming methods joined the Information and Communication Technology (ICT) world by evolving into Agriculture 3.0 in parallel to Industry 2.0 (second industrial revolution, 20th century) and Industry 3.0 that created fields such as embedded systems, software engineering, and renewable energy. Finally, the farming field is evolving from its current state (Agriculture 3.0) to its future shape (Agriculture 4.0) thanks to the ongoing fourth industrial revolution (Industry 4.0). This present evolution is mainly characterized by the fusion of many emerging technologies such as the Internet of Things (IoT), advanced electronics and robotics, big data, Artificial Intelligence (AI), and blockchain technologies (see Figure 1.1). Agriculture 4.0 (A4) is defined as a sustainable and intelligent industrial agriculture that is achieved through real-time variable fine-grained collection, processing, and analyzing of spatio-temporal data from all aspects of the agricultural industry. This new agriculture ecosystem is thus characterized by real-time farm management, a high degree of automation, and data-driven intelligent decision-making that can greatly improve productivity, food supply-chain efficiency, food safety, and the usage efficiency of natural and renewable resources ([Ye et al., 2021]).

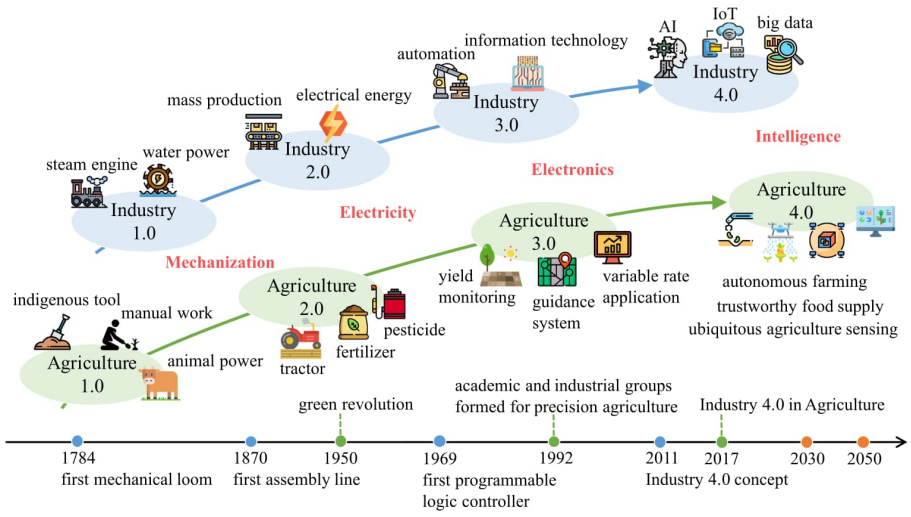

Figure 1.1: Parallel evolution of the industry and the agriculture according to the human technological advancements. (Source: [Ye et al., 2021])

Agriculture 4.0 is at the core of numbers of recent concerns. Being for environmental reasons, food availability necessity, worker safety conditions, phenotype and genotype research, machine learning systems developments, data and network management evolution, or human health improvement, it is one of the most studied vectors for improvements. From open-field cultivation to Controlled Environment Agriculture (CEA), this new agricultural paradigm impacts all farming techniques. Moreover, one of them stands out thanks to its trade-offs equilibrium and thus potentially is the most interesting A4 system: the Plant Factory (PF) and its successor the Smart Plant Factory (SPF). To understand why, here are four of the nine expected ultimate functions of SPFs from the book that defined this concept, titled "Smart Plant Factory: The Next Generation Indoor Vertical Farms" [Kozai, 2018]:

1. Contribution to solve the food, resource, and environment trilemma at the personal, local, regional, national, and global levels.
2. Contribution to improve the quality of life physically, mentally, and spiritually, in addition to food security including stable supply of nutritious and healthy foods.
3. Energy-autonomous, ecologically sustainable, and economically viable SPFs that achieve the highest production yield and quality with minimum resource consumption and maximum use of solar, biomass, fluid, thermal, and mechanical energy, resulting in minimum production costs and waste emission.
4. SPFs consisting of Cultivation System Modules (CSMs) networked with each other and open to most SPF users via the Internet. Standardization of CSM hardware and software sub-units to fit diverse types of CSMs.

As it can be understood from the first expected ultimate function in the list above, solving the environment trilemma (see Figure 1.2) is the reason why this agricultural revolution is at the core of numbers of recent concerns. This three-sided dilemma raises when the food plants production aims at high yield and high quality while wanting to preserve the cultivation environment and consuming the minimum quantity of resources. Currently, this trilemma has not reached an equilibrium and its undesirable alternatives thus prevail: unstable food-supply, resources depletion, and environment degradation ([Kozai, 2018]).

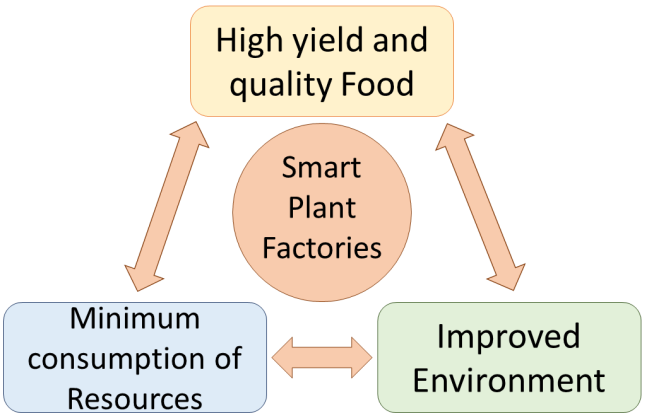

Figure 1.2: Resources-Food-Environment trilemma representation. (Original source: [Kozai, 2018])

Indeed, this same reference book states that "to help to solve this trilemma, transdisciplinary methodologies based on new concepts need to be developed by which the yield and quality of food are substantially improved with less resource consumption and environmental degradation compared to current plant production systems." [Kozai, 2018]. Agriculture has always been a

multi-disciplinary field (e.g., biology, agronomy, chemistry, etc.) but the consecutive industrial revolutions have brought various levels of engineering into the fields of interest. Moreover, the current agricultural paradigm shift is progressively integrating more ICT software and hardware ([Araújo et al., 2021], see Figure 1.3).

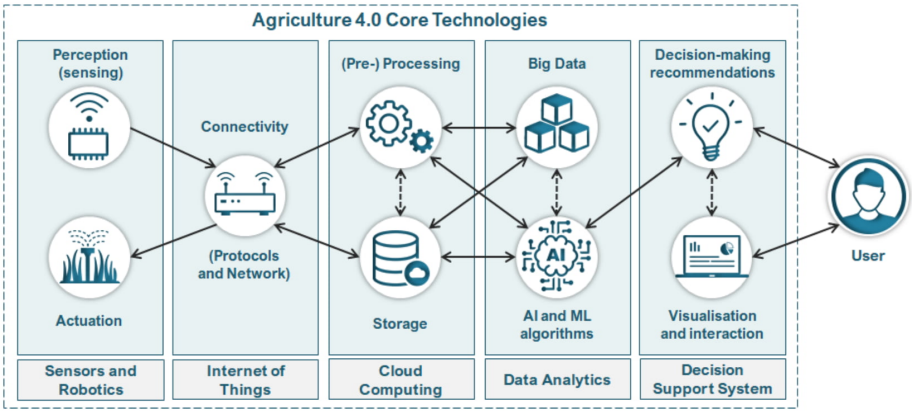

Figure 1.3: Agriculture 4.0 core ICT technologies and their interactions. (Source: [Araújo et al., 2021])

These SPFs are the main topic throughout this entire report. More precisely, the Automation and Intelligence Platform (AIP) that is used to build these structures is the center of interest of this work. This study choice was made thanks to the potential improvements these new technologies (e.g., big data, AI, IoT, etc.) can bring to the agriculture sector by solving many currently faced challenges ([Araújo et al., 2021], see Figures 1.4 & 1.5).

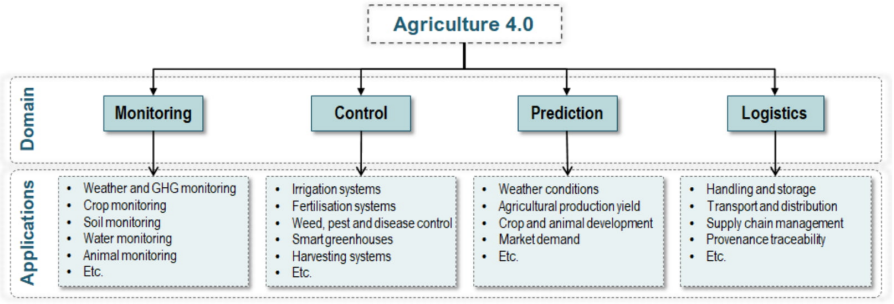

Figure 1.4: The Agriculture 4.0 core ICT technologies and their potential usages. (Source: [Araújo et al., 2021])

Given the wide scope of A4, this work cannot possibly cover each and every domain of research within SPFs. The field of interest that is the AIP for SPFs was also chosen because of the lack of scientific literature concerning this topic. In fact, the SPF world is dominated by enterprises that obfuscate their technical developments and only sell the final products or developed services. Fortunately, some scientific papers about SPF can be found but their main field of interest are often much more focused than enterprise projects. Most reports do not give many technical details about the implemented and used AIP. Moreover, some hobbyist works covering the development of such a platform are available but often suffer from a lack of scientific rigor.

More precisely, this master thesis aims at studying, developing, and testing a new AIP for SPFs. This hardware and software development should be able to accommodate most functions

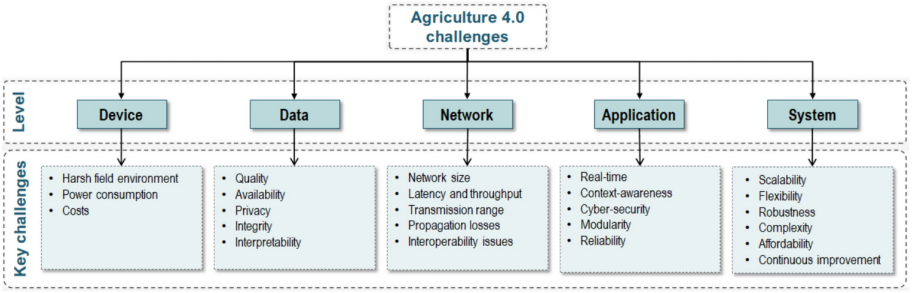

Figure 1.5: The Agriculture 4.0 key challenges. (Source: [Araújo et al., 2021])

needed by SPFs and thus improve the currently available solutions. To do so, the consumer-grade technologies are analyzed and the scientific, enterprise, and hobbyist literature are reviewed. Moreover, a Minimum Viable Product (MVP) of the control software and a Proof Of Concept (POC) of the monitoring hardware are developed.

The organization of this work is described hereafter. A theoretical reminder is firstly given in the second chapter of this work. Its first sections introduce important agricultural concepts and its last section reviews the State Of The Art developments from the various literature sources. With all the preliminary information laid down, the third chapter introduces the positioning, the objectives, and the methodology of this master thesis. Following these explanations, the fourth chapter describes the concepts used throughout the hardware and software developments while also reviewing the literature. Then, the three next chapters present the hardware developments, the software implementations, and the achieved results. Finally, a retrospective discussion, a perspective analysis, and a conclusion end this report.

Chapter 2

Background

Since the field of interest of this master thesis is multisectoral, this chapter acts both as a theory reminder and a clarification of the concepts that are involved in this project. The modern agriculture sector is in fact composed of a multitude of sub-fields and regroups various farming techniques. The noun "agriculture" can be defined as "the science, art, or practice of cultivating the soil, producing crops, and raising livestock" [Merriam-Webster, 2022a]. In contrast to conventional agriculture, modern-day farming techniques use technological advances to improve their production. More specifically, this report is interested in the fields of Controlled Environment Agriculture (CEA), Hydroponics System (HS), and Vertical Farming (VF). These three paradigms are modern agriculture sub-fields which have been recently improved thanks to Information and Communication Technology (ICT) fields. Moreover, they have been merged together under the concept of Plant Factory (PF) and Smart Plant Factory (SPF) which inherently incorporate the usage of ICT.

These farming techniques (and other ones, being similar or very different) all aim at being more efficient (with various meanings, e.g., economically, environmentally, etc.). However, this report is a dissertation in computer science oriented towards intelligent systems. Thus, it means that discussions are mainly focused on the ICT side of modern agriculture. That being said, it is still important to emphasize the benefits, disadvantages, and current research of this modern-day agronomic concept that are PFs. Additionally, the three aforementioned paradigms are also described to better understand the scope of modern agriculture systems builds with CEA, HS, and VF in mind.

2.1 Controlled environment agriculture

Controlled Environment Agriculture (CEA) is the first farming paradigm that this thesis uses. The acronym is pretty much self explanatory but, to be clearer, CEA regroups all the plants growing techniques for which some part of the culture environment are human-controlled. As explained in the article [Shamshiri et al., 2018], the most basic form of CEA that are the greenhouses have evolved to become complex systems that manage multiple aspects of the growing environment. The original CEA systems only controlled a few environmental parameters such as the air temperature, water for irrigation, or sunlight exposure. Nowadays, in addition to high-tech greenhouses (see Figure 2.1), advanced CEA systems are build in shipping containers or included inside modern city-buildings.

As stated in the same paper, "These structures [the CEA systems] use natural or artificial light within which optimum growth conditions are intended to achieve for producing horticultural crops, or for plant research programs. They also offer greater predictability, reduce the cost of production and increase crop yields." [Shamshiri et al., 2018]. Moreover, they can offer year-round production, better space usage, and protection for the plants against weather problems or fauna

and flora wildlife ([Duston, 2017]).

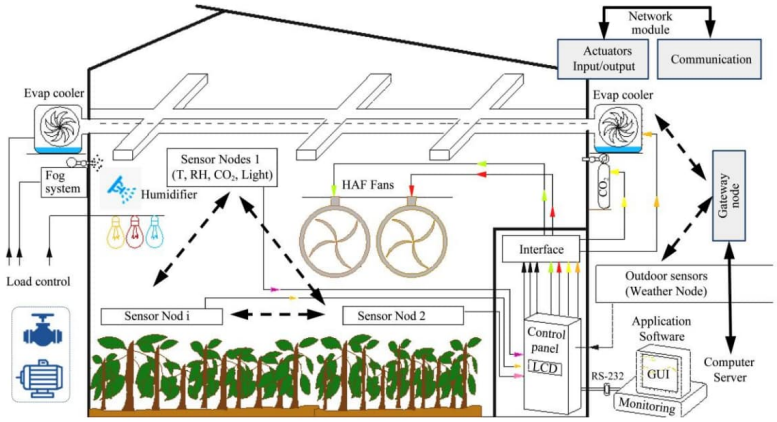

Figure 2.1: Schematic representation of a State Of The Art greenhouse. (Source: [Shamshiri et al., 2018])

Unfortunately, these infrastructures are not challenge-free. Alongside economical issues, due to building costs (e.g., covering material) and running costs (e.g., light and temperature control or irrigation systems), the main concerns are about the number, the variety, the quantity, and the load of grown crops compared to open-field cultivation ([Shamshiri et al., 2018]). Additionally, the large continuous energy requirements of such systems are often the most negative downside of CEA systems.

Finally, a study conducted in Detroit (USA) in 2017 [Duston, 2017] studies the sustainability and feasibility of CEA farms compared to traditional farming including multiple aspects. Their main conclusion is that "An overall restructuring of the agricultural system that uses traditional, organic, urban, and CEA in appropriate amounts and with attention to renewable energy is the key to a sustainable future, as well as changing distribution patterns so that products are transported mostly to local and regional areas.". As a side note, the energy footprint was pointed as the main challenge to overcome.

2.2 Hydroponics systems

The second farming paradigm concerned by this thesis is the Hydroponics System (HS). It can be defined as a plant cultivation technique that uses a nutrient solution with or without an inert medium that provides mechanical support. The roots can thus be directly immersed into the nutrient solution or grown into an inert medium (which is not soil) ([Textier, 2013]). The high-level fundamentals of HS are that the nutrient solution has to be kept at the best temperature while being oxygenated and the plants should receive through their root system the nutrients that they need. HSs can be classified as open or closed loop, and active or passive. Multiple system variants are in use: ebb and flow system, nutrient film technique, deep flow technique, drip system, deep water cultivation, etc. Moreover, hydroponics was derived into multiple variants: aero-hydroponics, aeroponics, aquaponics, bioponics, etc. The main differences between these variants are the way the nutrients are produced and how they are transported to the plant roots (see Figure 2.2). As a side note, the diverse system variants and technique variants can be combined, HSs are thus said to be very flexible.

The main strengths of an HS are the control over the nutrients distribution, better plant-health that leads to a lower usage of pesticide, more yields and of better quality, and access to the

roots ([Textier, 2013]). Moreover, depending on the chosen technique and system, an HS can be more environmental friendly, uses less resources, and creates agricultural opportunities in extreme weather conditions.

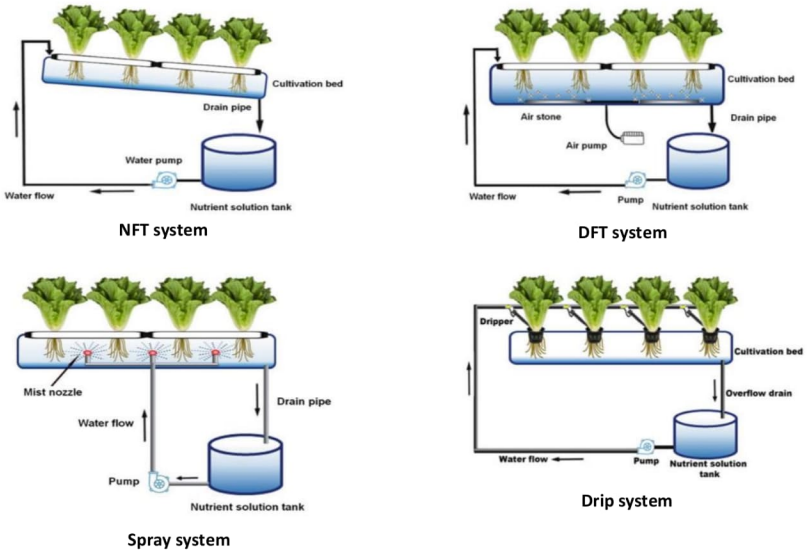

Figure 2.2: Schematic representation of various Hydroponics System variants: NFT corresponds to the nutrient film technique, DFT to the deep flow technique, and Spray to the aero-hydroponics technique. (Source: [Wildeman, 2020])

As usual, HSs have their limits: the lack of the buffering capacity from the medium (less error tolerance), the temperature management (harder in hotter climate zones), the specific design requirements needed by some plants (e.g., carrots), and the economical balance between cost and production ([Textier, 2013]). Solutions exist for all these shortcomings but they often increase the building and running cost of the HS.

Finally, multiple HSs are already in use or currently being tested. A study about a test-bed in Qatar was performed in the paper [Chowdhury et al., 2020]. They designed a system that could grow plants in this challenging climate while studying the price, power consumption, automation potential, and sustainability. Their main conclusion is: "This work presents significant opportunities for the people who live in the gulf region to produce food as per their requirements." [Chowdhury et al., 2020].

2.3 Vertical farming

The last required farming paradigm for this thesis is the Vertical Farming (VF). It is described by [Galanakis, 2021] as an indoor farming method that provides an unusual farming environment by not being restricted to a two dimensional space: plants are placed on grow tracks that are stacked level by level on top of one another. Such as for CEA and HS, VF systems have evolved overtime and they now come in various sizes and shapes (see Figure 2.3). The most basic ones (and often the smallest) are structures that can be hung on a wall or pyramid like modules. The most advanced ones (and often the largest) use CEA containers or shelves stacked up to multiple floors.

The major advantages of VF systems are their space efficiency and scalability thanks to their Three Dimensional (3D) shape. Moreover, their vertical component can also be used to allocate

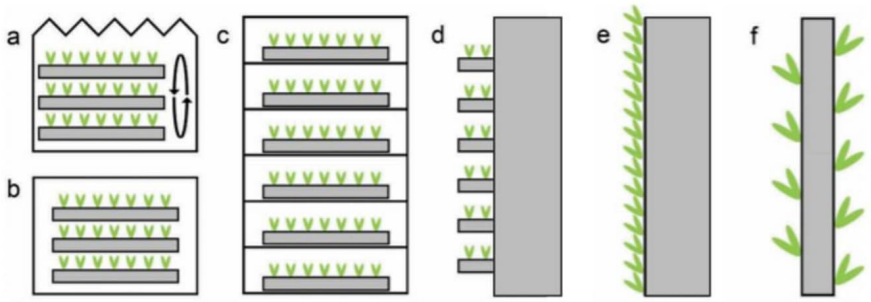

Figure 2.3: Schematic representation of various Vertical Farming variants: (a) stacked horizontal systems with level rotation, (b) stacked horizontal systems with controlled environment, (c) multi-floor towers, (d) balcony crop production, (e) green walls (that can be modified into a pyramid shape), and (f) cylindrical vertical growth units. (Source: [Wildeman, 2020])

new cultivation space in unused area (e.g., onto building walls).

Interestingly, their main shortcomings have evolved over time: "Most authors in the past have drawn the conclusion that the technology is not advanced enough to make a stable climate for the plants of which not enough is known, whereas most authors nowadays draw the conclusion that vertical farms are not economically feasible or profitable enough to stay alive without any large initial investments or funds during its user phase." [Wildeman, 2020]. Since VF and CEA are often used together, the current challenges of the VF field are very similar to the ones in the CEA field.

Finally, a study from 2020 about the feasibility of VF in Oklahoma City (USA) concluded that "Most of the claims stated in Chapter 2 [that is VF has a positive environmental, economical, social, and political impact] of this study are marked as correct, which is expected as they are based on some sort of logical thinking. However, as mentioned earlier, one claim that is used a lot on the top of the list of major vertical farm advantages, is in fact not true according to this study: The Land Footprint [because of the renewable energy production]." [Wildeman, 2020]. This conclusion shows that VF is not a perfect solution but rather a world of trade-offs.

2.4 Plant factories

Even if one could use each of them independently, the three agricultural paradigms that are CEA, HS, and VF are often used together. In fact, as said in the introduction of Chapter 2, this has led to the concept of Plant Factory (PF). These indoor vertical farms are described in the book from Toyoki Kozai "Plant factory - An indoor vertical farming system for efficient quality food production" [Kozai et al., 2019]. This description refers to PFs as thermally isolated and airtight plant production facilities that have a warehouse-like structure and in which multiple culture shelves are vertically stacked and irrigated with a nutrient solution.

When aggregated together, these farming techniques merge their pros without additional cons. Moreover, new advantages are even created and original ones are amplified. For example, herbicides and pesticides are not needed anymore, space usage can be heavily optimized, no water or nutrients are wasted, and the plants are fully isolated from the extreme and inconvenient weather and weather changes. Finally, if well designed, a PF has other higher-level core advantages: it can be built anywhere because neither solar light nor soil are needed, high resource use efficiency (water, carbon dioxide, fertilizer, etc.) can be achieved with minimum emission of pollutants to the outside environment (see Figure 2.4), and produced vegetables have a longer shelf life ([Kozai et al., 2019]).

It is however important to note that the main challenge with this type of infrastructure is the

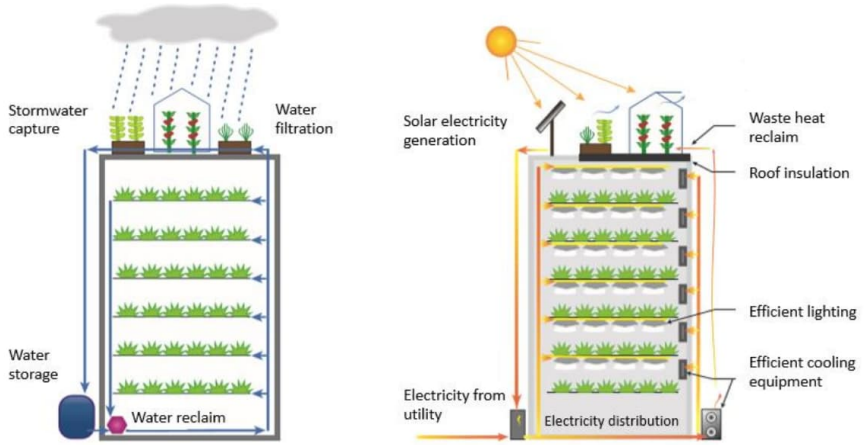

Figure 2.4: Example of a Plant Factory design with closed loop irrigation system and natural resources collection systems. (Original source: [Wildeman, 2020])

economical equilibrium because of their high building and running costs. Thankfully, it can be read in the same book that the hope to be able to reduce the effect of this major shortcoming is high: "The initial investment can be reduced significantly through better design. Fortunately, a positive aspect is that production costs are decreasing every year as operational and management experience accumulates. Electricity, labor, and materials (seeds, fertilizers, packing, delivery, etc.) account for similar proportions of the production costs. Among the total electricity consumption, lighting accounts for 70% - 80%, while air conditioning, pumps, and fans account for the remainder. There is great potential to reduce the cost of lighting by designing a more efficient lighting system. Other approaches to reduce production costs include increasing the number of vertical tiers, shortening the culture period by optimal environmental control, properly designing the production schedule to assure year-round production with no time loss, increasing planting density, and reducing production losses." [Kozai et al., 2019].

When looking in the literature discussed above, it can be seen that these three farming paradigms are often mixed implicitly. The book [Textier, 2013] about HS classifies the VF as an HS. The article [Chowdhury et al., 2020] about HS and VF has the final objective of implementing a CEA system while not explicitly using this paradigm. The thesis [Duston, 2017] about CEA uses hydroponics to implement their model of urban vertical farm and explicitly mentions the usage of VF. The paper [Shamshiri et al., 2018] about CEA sometimes refers PF as CEA (thus using HS and VF). Chapter 8 of book [Galanakis, 2021] about VF only talks about HS and present various CEA systems. And the thesis [Wildeman, 2020] about VF defines vertical farms as "any building that is designated to grow food inside of it with a controlled and monitored growing environment and climate, which is taller than one story and contains at least a semi-closed loop system of resource use." (thus referring to a PF).

In addition to the PF definition given above, some details can be provided. First of all, because this master thesis has not the objective of developing an entire PF, the described concept in this work regroups all the subfamilies of PF quoted in the reference book [Kozai et al., 2019] under the acronym "PFAL" (Plant Factory with Artificial Lighting). Secondly, this reference book and the other manuscripts from the same author (namely "Toyoki Kozai") are considered as the best references in the scientific community working in this research domain. And thirdly, with the same justification as for the first point, it can be said that many advanced theoretical concepts that are developed and studied in these references were not analyzed during this thesis writing.

The concept of PF is composed of six principal structural elements ([Kozai et al., 2006], [Kozai,

2007], see Figure 2.5):

1. A warehouse-like structure with a thermally well-insulated and almost airtight envelope.
2. A shelving system for multi-layer cultures where each culture bed can be equipped with various control systems (e.g., lighting, airflow, etc.).
3. A cooling/heating system (air conditioners/heat pumps) that is principally used for temperature regulation (to eliminate heat produced by lighting equipments) and air dehumidification (to reuse water vapor transpired by the plants).
4. An air quality control system principally used as a carbon dioxide delivery unit that enhances plants photosynthesis.
5. A nutrient solution delivery system used to irrigate plants.
6. An environmental control system that should at least include Potential of Hydrogen (pH) and Electrical Conductivity (EC) control units.

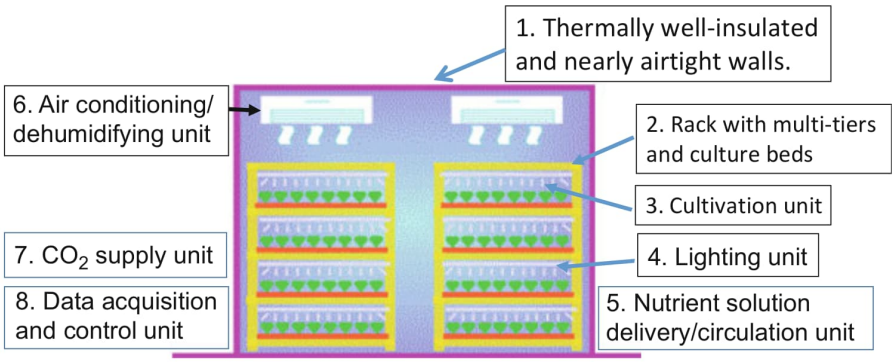

Figure 2.5: Schematic view of the Plant Factory concept with its six principal structural elements (n°2, 3, and 4 can be aggregated together). (Source: [Kozai, 2018])

Moreover, a well-designed PF should be built and operated to achieve these four main goals ([Kozai et al., 2006], [Kozai, 2007]):

1. Maximizing the usable and saleable amount of plants while using the minimum amount of resources.
2. Maintaining the highest "resources use efficiency" (a metric that defines the protocols to follow to use various resources as efficiently as possible).
3. Minimizing the pollutants released into the environment.
4. Minimizing costs while achieving the three previous goals.

Finally, a book from 2018 [Kozai, 2018] has studied the real-world adoption of PFs. It states that "As of September, 2018, the number of PFs for commercial production is roughly estimated at over 200 in Japan, about 100 in Taiwan, and over 500 in the world. The numbers of PFs in China, the USA, and Korea have been increasing significantly since 2015. During 2013–2017, one to several PFs were built in Singapore, Panama, Mongolia, Russia, France, Vietnam, the Netherlands, India, the UK, Malaysia, and a few Middle East countries including Dubai. Thailand and several other Southeast Asian countries are planning to build one or more PFs by around 2020." Moreover, even if the PF industry is still emerging, the author confidently says that the PF business is meant to continue to grow in the forthcoming years.

2.5 State of the art

The current State Of The Art (SOTA) concept is the Smart Plant Factory (SPF). This agricultural concept is described in the book [Kozai, 2018] as an intelligent or cognitive computing Plant Factory (PF) that has the ability to take decisions and solve encountered problems by itself (without human intervention). To have such an SPF, the factory should be able to adapt its behaviors by learning from its experience, not be totally dependent from people and instructions they give, and be able to respond to unexpected/unanticipated events. As of today, there is no known SPF in activity but it is important to keep in mind that the evolution in this sector is pretty quick. In fact, the PF research and development sector is only half a century old and the current PFs are considered as the fourth generation which has stated from around 2020 (and the SPFs are expected to be the fifth wave in the coming years, see Figure 2.6).

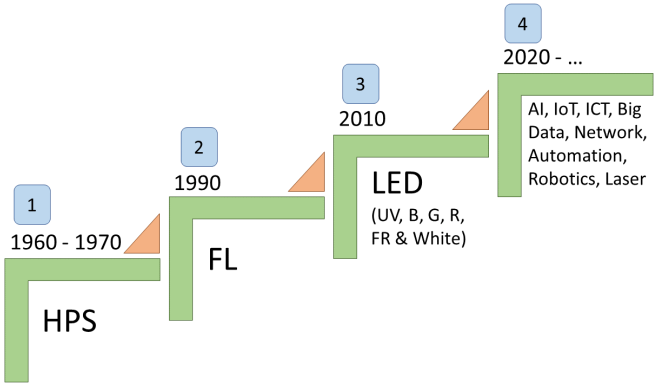

Figure 2.6: Evolution of the Plant Factory concept from its beginning with high-pressure sodium (HPS) or fluorescent lamps (FL) to nowadays with light emitting diodes (LED) with various spectrum (UV, R, G, B, FR, WW, CW) and tomorrow with Smart Plant Factory. (Original source: [Kozai, 2018])

Even if there is currently no SPF as presently defined, the current PFs are considered as smarter and smarter every years. The current installations are considered smart regarding their energy usage (e.g., off-hours or renewable energy usage), the lighting systems (e.g., adaptative lighting according to the grown plants and their requirements), the space usage (e.g., vertical and horizontal space optimizations), the resources usage (e.g., closed loop water and air systems), etc. In the remaining part of this section, the SOTA projects and literature about the ICT related fields around PF are reviewed.

The SOTA is currently shared by companies and scientists. In fact, multiple companies have started the development of promising systems and various scientific teams have produced interesting developments. Starting from the SOTA in the enterprise world, here is a non-exhaustive list of companies that have developed products or services linked to the SPF world:

- Intelligent Growth Solutions[1] has developed a CEA system which is called the TCEA platform (Totally Controlled Environment Agriculture) that is composed of growth towers. The focus is oriented towards scalability and profits but some automation and monitoring are available. No details about the used hardware and software are given an no compatibility list for custom growth tower is available. The proposed solution can be used to build a PF but no cognitive computation seems to be available.
- Urban Crops Solutions[2] has developed a complete PF solution composed of a modular factory

[1]Intelligent Growth Solutions website: `https://www.intelligentgrowthsolutions.com/`
[2]Urban Crops Solutions website: `https://urbancropsolutions.com/`

design and a dedicated software to control and monitor the crops. This all-in-one system does not offer any hardware nor software customization and no details about the used technologies could be found on their web pages. The developed PF does not offer any self-automation nor automatic adaptation, it is thus not a SPF.

- Cubic Farm[3] has developed two automated CEA VF HS: the CubicFarm System and the HydroGreen Grow System. The first one targets human food production while the second one aims at animals feeding. Both are heavily automated with moving trays or dual watering systems. Those systems could be used to build a PF but nothing is said about SPF future extensions.

- Cultinova[4] has developed an advanced, complete, and modular monitoring and automation software. They also propose hardware and growing system guidance. The system is open to custom hardware developments and tailored software extensions. The monitoring and automation system is compatible with all classical irrigation, nutrient, and environment sensors and actuators. This software solution is used to build PFs and could be extended for SPFs.

- Johnson Controls[5] has developed a complete suite of connected solutions for reliable building automation that is called OpenBlue. This high-level suite is a great development platform for PFs reliable automation. Custom hardware or software are not supported and no particular information is given for the A4 sector.

- Goponic[6] has developed a modular and scalable hydroponic gutter that can be used in all HS. This growing tray is compatible with the VF paradigm and it is used to build PFs. It does not provide any hardware automation or monitoring but can be combined with any additional software of interest.

- Atlas Scientific[7] has developed a wide range of hardware sensors and boards oriented towards environmental robotics that can be used to design a complete hydroponic control system. They provide hardware guidance and basic source code for customer grade hardware. The proposed solutions are not meant to be used for PFs but are oriented towards scientific installations and SPFs scientific developments.

- etc.

Within the scientific community, some teams have developed small-scale PFs with varying functionalities and different levels of monitoring and automation capabilities. No full-fledged SPFs development was found but some of the presented systems could be extended to support smarter automation systems. Here is a non-exhaustive list of interesting complete small-scale PFs developed in the recent years (see Figures 2.7 and 2.8):

- [Van et al., 2019] and [Chen et al., 2019] respectively called "PlantTalk" and "AgriTalk" present a smartphone-based intelligent HS plant box and a monitoring software with automation for a PF. These papers are based on the initial scientific developments that are "SensorTalk" [Lin et al., 2019] and "IoTtalk" [Lin et al., 2017]. These scientific articles are complementary to each other and present an overview of the software automation possibilities. Moreover, the used hardware is not closed to custom solutions.

- [Chowdhury et al., 2020] presents a container based PF developed as a test-bed in Qatar. The used hardware architecture and electronic circuits are precisely described. The developed system is tested thanks to the monitoring software and the basic automation allows the system to perform correctly. This test-bed could be used as a base for more advanced PFs and SPFs projects and is thus a good example for these developments.

[3]Cubic Farm website: `https://cubicfarms.com/`
[4]Cultinova website: `http://www.cultinova.com/`
[5]Johnson Controls website: `https://www.johnsoncontrols.com/`
[6]Goponic website: `https://goponic.fr/`
[7]Atlas Scientific website: `https://atlas-scientific.com/`

- [Lukito and Lukito, 2019] is a small-scale development of an IoT HS build with customer electronics and with a custom monitoring software and interface. Neither the used hardware nor the developed growing system are shown but the software architecture is explained. This work shows another hardware build that solves the same problems as the ones presented above.
- [Lakhiar et al., 2018] reviews the SOTA aeroponics (a sub-field of hydroponics). It presents the current systems, their components, their functions, as well as their current advantages and disadvantages. Aeroponics is one of the hardest but most effective systems to deliver nutrients to plant roots.
- [Abdelouahid et al., 2020a] presents a phytotron which is compatible with smart-home automation solutions. This CEA system is fairly different from the other in this list but incorporates the same functionalities. This phytotron is oriented toward scientific laboratories usage but its core components could be reused to develop a PF system.
- [Eduardo et al., 2017] shares the development of an all-in-one CEA system with an automatically regulated HS and a monitoring system. This build is well suited for home usage and its extension towards building a PF is hard but feasible.
- etc.

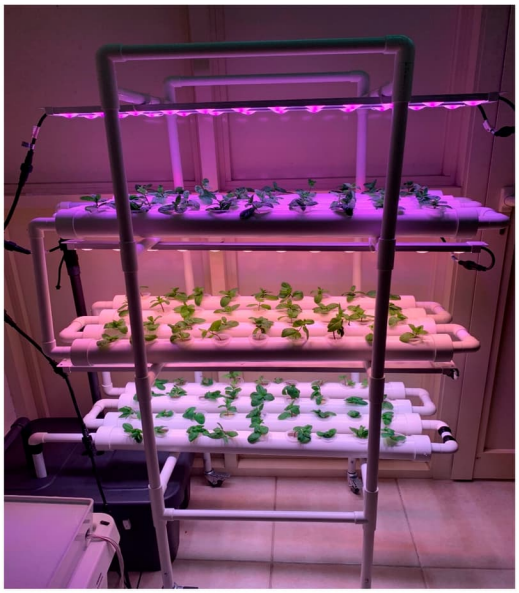

Figure 2.7: Example of small-scale Plant Factory that is developed and used by scientists. (Source: [Chowdhury et al., 2020])

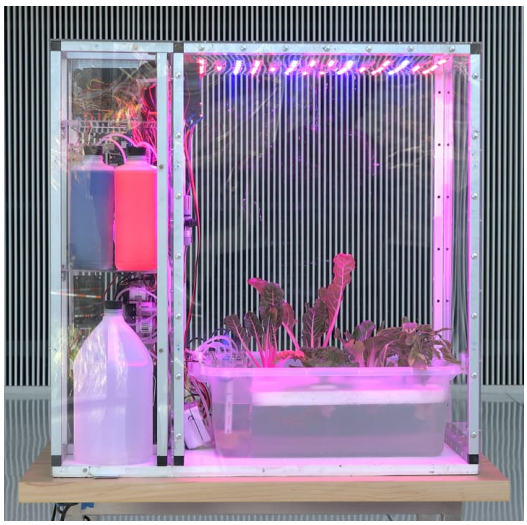

Figure 2.8: Example of an all-in-one CEA systems that is developed and used by scientists. (Source: [Eduardo et al., 2017])

Finally, the SOTA also concerns some specific components of PFs and SPFs. In this regard, the scientific community is heavily active and predominates over the enterprise world. Many ultra specialized, highly efficient, and very complex systems are studied in various sectors used in A4. Here is a non-exhaustive list of scientific reports and reviews describing these works:

- [Park and Kim, 2021] implements a computer vision AI to monitor strawberry production while predicting harvesting timings. This approach is based on container virtualization and cloud computing for executing the prediction AI. In opposition to the approach used in [Debauche et al., 2020], this work shows one path to integrate ML workloads into PFs by setting-up Linux containers (LXD) for each server (e.g., the DataBase server, the ML server, etc.).

- [Jaenuri et al., 2021] is an implementation of an automatic canopy space expander system that could be used to optimize space usage in PFs. The presented hardware solution is compatible with classic HS and the software flowchart is described. The authors have used classic computer vision algorithms to detect and analyze the plants canopy. This work presents a hardware system that could be used to improve current PFs and that could be easily combined with other systems.

- [Cambra et al., 2018] presents an automatic pH auto-calibration system. The intelligence of this system is based on a flowchart that is described in the paper. The used hardware and IoT network are also explained in the article. This solution could be combined with other systems of this list to be implemented into a PF to make it smarter (e.g., by having an AIP that allows auto calibration when unexpected measurements are observed).

- [Lin et al., 2017] is the original paper in which a platform for IoT device interactions is developed. This powerful tool provides a web interface and a custom automation system that were both used in the three follow-up papers "SensorTalk" [Lin et al., 2019], "PlantTalk" [Van et al., 2019], and "AgriTalk" [Chen et al., 2019] (newer articles from the same system have came up since 2019 but their main focus is not linked to this work: objects recognition and smart homes). This work showcases a powerful automation software that can be highly customized to implement better automation. Unfortunately, nothing is said about smarter algorithms and cognitive computing.

- [Verma and Gawade, 2021] presents a Machine Learning (ML) approach for predicting the nutrients uptake in an HS. The proposed system can be used to give a balanced supply of nutrients required by plants for optimizing their growth. The paper does not describe the used hardware, software, nor the implemented ML algorithms but it presents the software flowchart. This type of system could be used in the AIP of an SPF to gradually learn and improve the nutrients supply.

- [Debauche et al., 2022] presents a new edge computing architecture for A4 that is specialized for multimedia-data management. This approach has the objective of solving two problems that are congestion and computing time of IoT devices while ensuring data privacy protection and system real-time responsiveness. This article is very detailed regarding the edge computing architecture and this kind of systems will be needed in SPFs to transform low value raw data into high value synthesized data.

- [Debauche et al., 2020] is an AI implementation of a Poultry monitoring system that uses edge computing. In opposition to the system presented in [Park and Kim, 2021], this real world combination of IoT and edge computing shows what type of cognitive computing could be developed inside IoT for implementing smart culture trays in SPFs. Moreover, the paper presents some useful hardware and a great literature review was conducted.

- [Penghui et al., 2022] presents an AI deep learning approach that uses an artificial neural network to detect cherry tomato fruit and stem. This paper does not present the integration of the visual recognition system into a PF but it studies the required hardware camera needed for such systems. This ML algorithm could easily be merged with the integration studied in [Debauche et al., 2020] or [Park and Kim, 2021].

- [Kim et al., 2022] review a high-level ML approach for predicting plant growth. The paper is not detailed but it states interesting results. This type of cognitive computing has a huge potential and could allow PFs to become SPFs if the needed AIP is developed.

- etc.

Chapter 3

Methodology

Following the background information given in Chapter 2, this part of the report gives concrete explanations about the master thesis positioning, the project developments, the objectives, and the methodology.

3.1 Thesis positioning

As it has been stated in the introduction (Chapter 1), there are three reasons why the redesign of the Automation and Intelligence Platform (AIP) for Smart Plant Factories (SPFs) was chosen. Firstly, because it is heavily linked to a master in computer science with a professional focus on intelligent systems. Secondly, for the reason that there are currently no SPF in activity because of the lack of a proper AIP alongside other cognitive computing systems. Thirdly because of the lack of scientific literature concerning this topic.

About this last reason, as it can be noticed in the SOTA analysis (Section 2.5), the field is dominated by enterprises developments when looking for the completely featured systems literature. However, zero or very few technical details are made available by these companies (because their developments are not open-source). Some high-level details can sometimes be found in conference reports or in the grey literature but they are not accurate enough to understand the underlying developments produced by companies.

Concerning domain-specific SOTA developments, the literature is dominated by scientists. Unfortunately, once again, only few technical details and implementation explanations about the AIP are given in the various scientific papers. For example, among the reviewed scientific articles in Section 2.5, almost none of them gives access to their source code (neither for the developed ML algorithms nor for the web interfaces). Moreover, only shallow ICT details are given even when looking at the three papers that implement a fully-featured growing system. The "AgriTalk" article ([Chen et al., 2019]) offers an in-depth review on the configuration and possibilities of their dashboard but says nothing about the way that it is implemented. The Qatar test-bed article ([Chowdhury et al., 2020]) also complains about the lack of technical details in the articles that it reviews but does not give any information about the languages, frameworks, and technologies used in its solution (the authors however give the formulas that they used to automate the HS control). The small-scale growing system article [Lukito and Lukito, 2019] is mildly different because it mentions some programming languages, explains the used DB technologies, and states the used hardware communication ports.

When looking for some of these technical details (e.g., source code, electronic architecture, etc.) and implementation examples, one needs to fall-back into the "maker space" in which hobbyists like to share their homemade developments. The best example of such a hobbyist work is the project

"Arduino Controlled Smart Hydroponic Modular System"[1] built by Innovart Studio. This project is given with a step-by-step tutorial, 3D illustrations, a part-list, an electric schematic, and the source code. It implements a VF HS and thus can be compared to a small-scale PF if some environment variables are controlled (e.g., if the system is used indoor). Another good example is the project "GardenPi"[2] built by Neptune. This project is an aquaponics (a sub-field of hydroponics) system that is given with a part-list, an electric schematic, a demonstration, and the source code. This second example does not develop a PF but uses the aquaponics system to irrigate outdoor plants. Fortunately, it could easily be adapted to be used in a CEA system. A third project that could be adapted to the need of this work is a pool monitoring[3] project built by Dominic at HydroPi. This overview comes with a part-list and the needed source code. This last project is not focused onto agriculture but the developed monitoring system is very similar to HS control software.

However, it cannot be said that there is absolutely no scientific literature covering projects similar to the three aforementioned hobbyist works. The already presented article [Abdelouahid et al., 2020a] presents a phytotron that is a controlled environment culture chambers used by researchers to study the impact of environmental parameters on plants. This implementation is open-source (a Fritzing[4] project is given) and it is compatible with customer-grade smart-home infrastructure (i.e., Home Assistant and OpenHAB). The other reviewed earlier article [Eduardo et al., 2017] also implements an all-in-one CEA solution that is called "the OpenAg Personal Food Computer". It also is open-source (a GitHub repository is linked) and the 3D plans are given with the electronic part list. The paper [Cabaccan et al., 2017] presents a nodes-based wireless sensor network that can be used for monitoring multiple CEA systems simultaneously. This infrastructure is not open-source but the electrical connections are described. This last scientific development [Abdelouahid et al., 2020b] was not developed for the SPF field but is also comparable to the hobbyist developments while being more scientifically rigorous. It shows the result of the development of a monitoring system for IoT based artificial cavities for birds nests. This system was developed by the same authors than [Abdelouahid et al., 2020a] and is thus very similar: it is open-source (a GitHub repository is linked) and it is compatible with customer-grade smart-home infrastructure.

Unfortunately, among these few scientific reports, none provides an in-depth review of the implemented monitoring and control platform. In fact, the common structure of these projects is composed of three parts: a cloud and network infrastructure review, a hardware presentation, and a software overview. The first part is often the more detailed one, it is also sometimes the core of some studies. The hardware presentation is most of the time illustrated with electric schematics and part lists but its shortcomings are rarely discussed. Finally, the software overview is often the less detailed part because only a few papers present a high-level view of their systems while not talking about programming framework nor code architecture and their respective use-cases. Moreover, most of these works could be extended or used to setup a PF but it cannot be said that they implement an AIP because of their technical limitations (being hardware or software). This is why the final positioning of this master thesis is dictated by these three reasons. The main goal is to review and build an Automation and Intelligence Platform (AIP) with the same scientific rigor than the scientific literature while using technologies similar as in the hobbyist world and while trying to implement a system that can be used the same way than enterprise solutions and scientific projects.

3.2 Developments objectives

From the project objectives covered in the introduction (Chapter 1), it can be noticed that this master thesis aims at studying, developing, and testing a new Automation and Intelligence Platform (AIP) for Smart Plant Factories (SPFs). Before going further, the AIP concept should be explained. The word "platform" can have multiple meanings according to the application field. For

[1] Smart Hydroponics link: `https://www.instructables.com/Arduino-Controlled-Smart-Hydroponic-Modular-System/`
[2] GardenPi link: `https://www.hackster.io/user3424878278/gardenpi-powered-by-neptune-c0a691`
[3] Pool monitoring project link: `https://myhydropi.com/overview`
[4] Fritzing website: `https://fritzing.org/`

example, in the computing world, it is defined as "an application or a website that serves as a base from which a service is provided" [Merriam-Webster, 2022b] and, in the automotive industry, as "a basic pattern upon which an automobile or class of automobiles is constructed" [Merriam-Webster, 2022b]. In the context of this work, the platform definition is a mix of these two examples: it is a basic Information and Communication Technology (ICT) hardware pattern and software structure upon which ICT systems that serve as a base for various services are built. Moreover, for the PFs and SPFs, the concerned services are the automation algorithms execution and the cognitive computing developments. As it can thus be deduced from the aforementioned definition, the hardware and software developments should be able to accommodate most functions needed by an SPF. By doing so, the main objective of this master thesis is to improve the currently available solutions regarding both the available services and in terms of hardware and software descriptions. To do so, some consumer-grade technologies are analyzed and the scientific, enterprise, and hobbyist literature are reviewed.

With a higher level view, the main goals of the work are to build a Minimum Viable Product (MVP) of the control software and a Proof Of Concept (POC) of the monitoring hardware. To be clearer, an MVP is a product version in which the critical features are usable and that is used to plan the future product developments. Similarly, a POC is a development of an idea or a method that demonstrates the principles and the feasibility of a concept. These two concepts are used to verify the real world theoretical and practical potential of developments. In the case of this master thesis project, the MVP of the control software and the POC of the monitoring hardware are used to demonstrate the developed AIP prototype and to assess its strengths and weaknesses.

The project reviewed in this report is multi-disciplinary: some hardware components and electric circuits were designed, all details of the software pieces were studied (e.g., programming language, development framework, network communications protocol, etc.), and some agricultural systems have been analyzed to develop a test bed. Each one of these fields of interest have an impact on the developed AIP prototype. Alongside the AIP low-level objectives, the multi-disciplinary aspect of the developed system is reviewed later in this work in Section 4.2.

3.3 Methodology details

The methodology followed during this master thesis could be described as prototype based. It can be defined as a development model in which a prototype is built, tested, and then reworked. This procedure is repeated (if needed) until an acceptable prototype is achieved. This type of methodology is perfectly compatible with the MVP and POC concepts.

The steps involved into the development of this project can be summarized as five main milestones:

1. Reviewing the high level concepts defined by scientists in the SPF field.
2. Analyzing the available literature to understand the current research subjects and the presently build AIPs.
3. Building a POC of the monitoring hardware to support the AIP software development.
4. Implementing a MVP of the control software that is composed of an Electronic Management System (EMS) linked to a Management Dashboard (MD).
5. Regrouping these two developments to assess the built AIP.

These five steps are discussed in this document: Step 1 is developed in Chapter 2 (all sections) and in Chapter 4 (Sections 4.1 and 4.2); the second step is considered in the same chapters in Sections 2.5, 4.3, and 4.4; Chapter 5 is devoted to Step 3; the fourth step is addressed in Chapter 6; and the last step is developed in Chapter 7 and reviewed in the Chapter 8.

Chapter 4

Concepts

This dissertation project is based on two mains theoretical concepts that exist to structure the monitoring and control systems of SPFs. These two theoretical concepts were already mentioned in previous chapters: the Cultivation System Module (CSM) and the Automation and Intelligence Platform (AIP). This part of the report is dedicated to formalize these concepts while also reviewing the technical side of the already presented literature. In fact, this technical review is performed to explain the hardware and software choices made during the project realization.

4.1 Cultivation system module

The farming paradigm used and studied in this master thesis is the SPF concept. As said before, it gathers the HS, VF, and CEA agricultural paradigms. However, one key concept that was mentioned in the introduction (Chapter 1) is needed to define the SPFs: the Cultivation System Module (CSM). The CSM is a theoretical concept from Toyoki Kozai's book "Smart plant factory: the next generation indoor vertical farms" [Kozai, 2018]. This design describes the physical layout of the growing system and its relations with the hardware components and the software stack. The CSM concept can be detailed as follows: "The CSM is characterized by its properties of scalability (potential to be enlarged to accommodate an increase in number of CSMs and/or cultivation racks), controllability, adaptability, and so forth [...]. The CSM needs to be lightweight and simple in structure. Each CSM handles the low-level measurement, control, and information processing itself." [Kozai, 2018]. In other words, the CSM is the base component of the plant factory for growing vegetables that should be designed to be stacked, adapted, and improved.

This Cultivation System Module concept is pretty important even if the CSM considered in this thesis does not implement all the features explained in the original Toyoki Kozai's description. This is due to the fact that these master thesis objectives are to study the Automation and Intelligence Platform (AIP) of SPFs. However, all the non-implemented features are still totally compatible with future versions of the CSM thanks to its modular design (e.g., to add hardware sensors) and the developed AIP (e.g., to add monitoring software).

To be more accurate, here is a list of the CSM characteristics as it is described in the reference Toyoki Kozai's book. This list also shows which capabilities are currently supported by the CSM design developed for this master thesis project (✓ symbol) and which features could be added in future revisions of the growing system (• symbol).

- ✓ Scalability:
  - ✓ Modularity: make each component of the CSM as independent as possible.
  - ✓ Connectivity: stabilize and make reliable each and every connection within the CSM and between the CSMs.

✓ Upgradability: make possible step-wise improvements.

- Controllability:
  - ✓ Measurement: enable the measurements of various system parameters.
  - ✓ Storage: define a storage strategy for online and offline functioning.
  - ✓ Processing: create analyzing tools to make use of the collected data.
  - ✓ Control: allow the system to be controlled via the control/monitoring system.
  - • Phenotyping: phenotype the plants thanks to the control/monitoring system.
- Adaptability:
  - ✓ Openness: study the openness of the security, standardized protocol, and application software interface.
  - ✓ Maintainability: enable easy repairs and replacements.
  - • Sustainability: assess the life-cycle of the CSM.
- Productivity:
  - • Production Costs: calculated as the product of the unit economic value ($/kg) and the resource element consumption per kg of production.
  - • Cost Performance: calculated by the ratio of sales (S) to the production cost (C), the S value can be calculated as the product of the economic value (U), the volume of produce (P), and (1.0-L) where L is the produce loss ratio.
  - • Payback Period: define the productivity by resource element and the resource element consumption per kg of product to study the break-even point.

As said in the beginning of this section, all the CSM features are not currently studied. For example, neither the lighting system nor the air-flow pattern were studied during the development of the proposed CSM. However, as mentioned before, the system was developed to be compatible with pretty much any future modification. When looking at the airflow design example, this modular system has the ability of being transformed into any type of CSM defined in the reference Toyoki Kozai's book (type A to D with their ramifications, see Figure 4.1). Moreover, the story is the same regarding the environmental factors monitoring, the resources supply rates control, the production rates computations, the plan phenotypic traits analysis, and the resource use efficiencies factors.

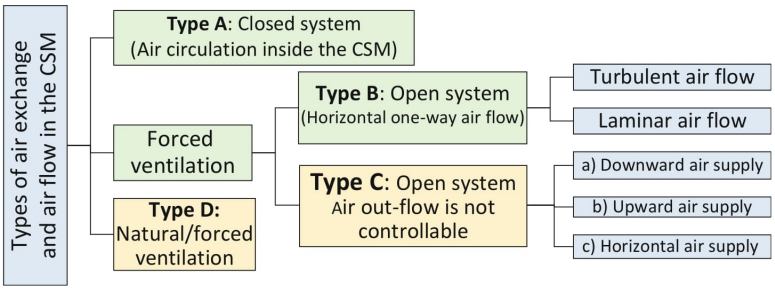

Figure 4.1: CSMs types given the air-flow pattern. (Source: [Kozai, 2018])

Finally, concerning the CSMs interactions and balance between cloud, local, and edge computing, the architecture defined in the reference Toyoki Kozai's book (see Figure 4.2) can be fully implemented by the AIP developed for this master thesis.

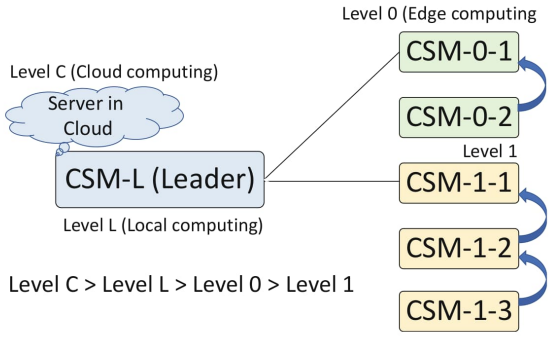

Figure 4.2: CSMs interconnections logic. (Source: [Kozai, 2018])

4.2 Automation and intelligence platform

To support this CSM concept, an ICT infrastructure has to be set up. For this dissertation, this ICT system is called the Automation and Intelligence Platform (AIP). It is composed of four elements: the electronics, the CSM, the SPF, and the human control access. These can be seen in Figure 4.3. This real-world organization impacts the hardware and software sides of the AIP. In fact, it will be seen later that the hardware and software logical organizations are very similar to this physical construction. Moreover, colors and names used in every universal modeling language style diagram of this report do correspond with each other. This was made to help understanding which elements are part of which other diagrams and vice-versa.

The objective of the AIP is to organize the communications and roles of each ICT module. To do so, an architecture has to be designed to make the system scalable and controllable. Moreover, the defined modules should know what their role is and with which other modules they should communicate. In the context of this master thesis, the design goal is to set-up an ICT infrastructure supporting the SPFs control system in which it should be easy to add automation and study cognitive algorithms while keeping the scalability and the controllability of a classic SPFs control system.

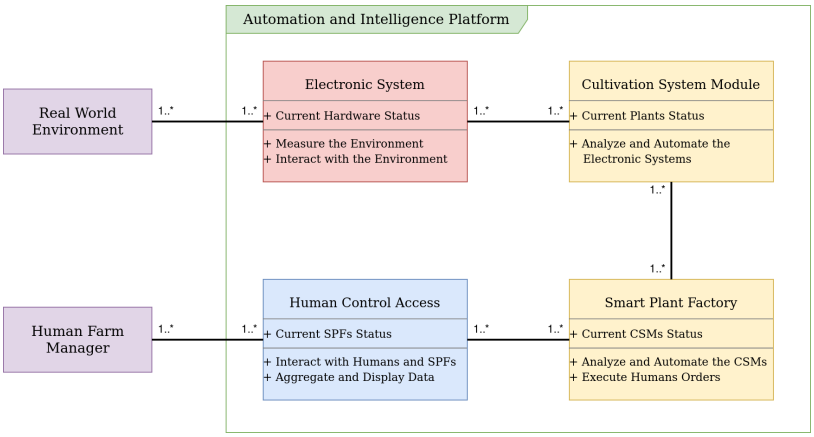

Figure 4.3: High level component diagram of the real-world SPF AIP. Each component is a conceptual element required in the real-world SPF AIP realization. These components are each composed of hardware and software parts: the yellow ones compose the Electronic Management System (EMS), the blue box is the management system (accessed via the Management Dashboard), the red component is the electronic assemblies, and the external elements are in purple.

To have a better understanding of the AIP concept, Chapter 25 of [Kozai et al., 2019] and Chapter 4 of the second book from the same author [Kozai, 2018] describe the three fundamental design elements of the AIP for SPFs: the information, the physical environment, and the spacial organization. These three elements are interdependent and the AIP has thus to be designed accordingly by studying each of the three elements. More precisely, on one hand, the information (collection, distribution, and analysis) system is dependent of the environment because its information sources (e.g., sensors, probes, etc.) are dependent of the environment. On the other hand, the information system is dependent of the spacial organization of the PF because it has to take into account this organization for analyzing the collected information. Then, the spacial organization is dependent of the information system to get accurate data and to correctly transmit movements. And the spacial organization is also dependent of the physical environment because its hardware is dependent of the physical environment. Finally, the physical environment is dependent of the two other fundamental elements to evolve in the best possible manner. As a side note, these interactions are represented in Figure 4.4.

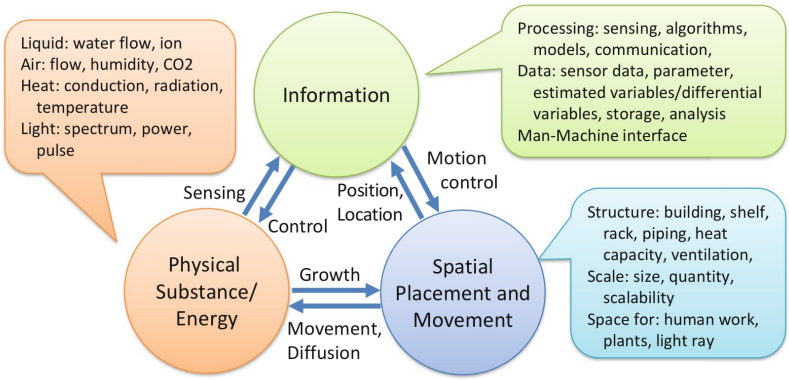

Figure 4.4: The three fundamental design elements of the AIP and their interdependence. (Source: [Kozai, 2018])

From the SOTA review (Section 2.5), the AIP definition (Section 3.2), and the description given above, the main low-level objectives of the AIP are:

- Interface the necessary hardware to monitor and control the physical environment.
- Collect, distribute, and store the physical environment monitoring and control data.
- Implement some decision algorithms and interact with them (being from the cognitive computing world or not).
- Provide an interface for human interventions (with data input/output capabilities).

Moreover, for this master thesis, some additional objectives were defined to improve upon the currently developed control and monitoring systems:

- Have a modular design in terms of hardware support, decision algorithms implementation, and software design.
- Be reliable, autonomous, and convenient to use.

4.3 Hardware review

To implement the hardware side of the AIP, some choices need to be made. This section reviews the hardware used in the literature, explains the advantages or drawbacks of each approach, and justifies the components choices made for the developed AIP. It is important to understand the scope of the hardware choices. In fact, these decisions have an impact on the software side of the AIP but also on the final capabilities of the AIP. Moreover, the made choices should also be compatible with all the low-level and high-level objectives of the AIP.

The next paragraphs are each focused on one hardware dilemma and each point is reviewed in three steps. To start, some theory is given about the concerned field of interest, then a review of the relevant literature is performed, and finally the choice made in the context of this work is justified. The topics that are analyzed hereafter are: the processing unit type, the computing board features, and the wanted sensors and actuators requirements.

Concerning the processing unit type, two possibilities are available: microcontrollers and microprocessors. Moreover, multiple families of each type of processing unit exist. For microcontrollers, these are: embedded microcontrollers, external memory microcontrollers, 8 bit microcontrollers, 16 bit microcontrollers, 32 bit microcontrollers, etc. Regarding microprocessors, these families are: complex instruction set microprocessors, application specific integrated circuit microprocessors, reduced instruction set microprocessors, digital signal microprocessors, etc. For the development of the hardware side of the AIP, the interest is focused on embedded microcontrollers and reduce instruction set microprocessors. In fact, the dilemma is on the choice between two different ecosystems: MicroController Units (MCUs) (e.g., Arduino[1]) and Single Board Computers (SBCs) (e.g., Rapsberry Pi[2]). A microprocessor can be described as a controlling unit of a microcomputer that is wrapped inside a small chip, performs arithmetic logical operations, and communicates with the other devices connected to it. In short, it is a single integrated circuit in which several computing functions are combined. On its side, a microcontroller is a chip optimized to control electronic devices that contains memory, processor, and programmable Input/Output (I/O). All these components are stored in a single integrated circuit which is dedicated to perform a particular task and execute one specific application. The main high-level differences (interesting in the context of this work) between these two computing unit are thus the processing power, the power consumption, and the functionalities. Microprocessors are often more powerful than microcontrollers while consuming more power but are compatible with higher efficiency systems (e.g., higher speed network and memory protocols, higher abstraction level programming language, etc.).

In the reviewed literature (Section 2.5), both processing units are used. Microcontrollers are used in the articles [Chowdhury et al., 2020] (Arduino Mega 2560), [Cambra et al., 2018] (Arduino Uno), [Lin et al., 2017] (STM-32), [Abdelouahid et al., 2020a] (ESP32-Wroom-32), and [Abdelouahid et al., 2020b] (ESP32-Wrover-32). Microprocessors are used in the articles [Jaenuri et al., 2021] (Raspberry Pi 1B+), [Debauche et al., 2022] (Raspberry Pi 4, NVidia Jetson Nano, and NVidia Xavier NX), [Eduardo et al., 2017] (Raspberry Pi 3B), and [Cabaccan et al., 2017] (Raspberry Pi 3B+). Additionally, some articles present hybrid systems that either use both type of processing units or cloud computing capabilities. It is the case of the articles [Van et al., 2019] (ESP8266-12F, ROHM IoT kit, and MediaTek LinkIt Smart 7688 Duo), [Park and Kim, 2021] (Arduino Mega 2560, Raspberry Pi 3B+, and GPU cloud computing), [Lukito and Lukito, 2019] (Raspberry Pi 1B and Wido 1), and [Debauche et al., 2020] (NVidia Jetson Nano and ESP32-Wroom-32). From these lists, a general trend can be understood: MCU are preferred for self-sufficient systems, SBC are used for more computing dependent systems, and hybrid approaches are created for more complex systems.

[1] Arduino website: `https://www.arduino.cc/`
[2] Raspberry Pi website: `https://www.raspberrypi.com/`

For the the hardware side of the AIP, two factors have to be taken into account: the networking and the computing power. As it will be seen in further chapters of this work, the developed AIP has a multi-node architecture and these nodes need to communicate over a local network. For that reason, all the MCU without wifi or ethernet supports are not ideal. Moreover, as it is also said later in this work, both the edge-computing and the cloud-computing paradigms compatibility are wanted. This thus forces the choice towards the SBCs. Moreover, the computing unit needs to be compatible with ML frameworks (e.g., Tensorflow[3]) and programming languages which offer the needed functionalities (e.g., Python[4]). This also forces the choice towards the SBCs. The final choice is thus to use the Raspberry Pi family because it satisfies all the requirements and because the implementation developed for this platform could easily be transferred to more powerful SBCs if needed (which is not the case if the initial developments are made for MCUs).

Regarding the the computing board features, the Raspberry Pi (RPI) family offers multiple options: the RPI 4, the RPI Zero 2(W), the RPI Compute Module 4, and the RPI Pico. However, the last one is not relevant because of the previously made choice seeing that the RPI Pico is an MCU. Moreover, the RPI Compute Module 4 is a computing board that lacks I/O ports and thus needs an I/O daughter board to be useful in this development. To avoid this trouble, this SBC is not considered as a viable alternative at this stage of the development. Moreover, the RPI 4 is exactly identical to the RPI Compute Module 4 but has all the needed I/O ports.

When analyzing the articles quoted above that use SBCs, only the B variant of the RPI family is used (the current RPI 4 is the successor of this variant). The choice may thus seem obvious but one additional fact needs to be taken into account: the RPI Zero 2(W) was released after the publication date of all these articles. Moreover, the predecessor of this board was not very convincing and that is the reason why it was not used in any of the previously seen developments.

For the current developments, the need for computing power is currently not extremely high. The main difference between the RPI 4 and RPI Zero 2(W) is their computing power and their I/O. Seeing that the I/O ports available RPI Zero 2(W) are sufficient for this project needs (i.e., only the GPIO ports are used) and that the computing power of both model is sufficient, the wifi variant of the RPI Zero 2W (Figure C.6) is selected for the development of the AIP. This choice can be observed in Figure 4.5.

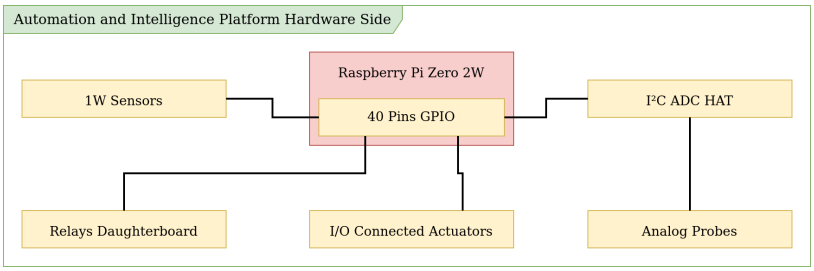

Figure 4.5: The hardware side of the AIP.

Concerning the wanted sensors and actuators requirements, the most important aspect is to be compatible with as many devices as possible. At first glance, the RPI platform is already a good match seeing that the available SBCs are compatible with most of the used communication protocols (e.g., I²C, UART, 1W, etc.). Moreover, in cases where the device protocol is not supported, an Hardware Attached on Top (HAT) or an external board can be used to interface this particular device.

[3]Tensorflow website: `https://www.tensorflow.org/`
[4]Python website: `https://www.python.org/`

When looking in the literature, the main trend that can be found is that most projects used customer grade sensors and actuators. As a matter of fact, the some devices used in the articles [Park and Kim, 2021], [Lukito and Lukito, 2019], [Debauche et al., 2020], [Abdelouahid et al., 2020a], and [Eduardo et al., 2017] are also used in this project.

The RPI Zero 2W choice thus still holds as the hardware platform for development of the AIP. This device fulfills all the needed requirements and is a great starting point to implement all the wanted functionalities of the AIP. This choice and more are discussed in Chapter 5 which reviews the hardware development of the AIP.

To end this section about the hardware choices, one last interesting point can be explained. In today's world, no major electronic platform is unusable. All the currently popular solutions are widely used and compatible with almost any wanted electronic device. For example, the chosen RPI platform should inhibit the use of 5V sensors but, with the help of a logic level converter, this limitation is non existent. The decisions taken in this section are obviously the result of deep research and are thus the best possible choices for the current development. However, at the time of writing, the Arduino Pro lineup was renewed and now contains the Protenta-x8 board that directly integrates a microcontroller and a microprocessor. This type of board is thus a new viable alternative to the use of hybrid system with both an Arduino and an RPI boards. The main concluding remark is that between popular platforms, none could have been a bad choice because the downside of each platform can be balanced by connecting some other kinds of components (e.g., HATs or shields, edge or cloud computing servers, etc.). Finally, when designing a hardware system, two dilemmas occurs: homogeneity or heterogeneity and general purpose or specialized. For each dilemma, both approaches have their pros and cons and none is really better than the other. For this work, the emphasis is put on the homogeneity and the general purpose of the system. This is thus valid for the hardware choices and it is wanted because the developed AIP is intended to evolve and is opened to improvements.

4.4 Software overview

As for the hardware side of the AIP, some choices need to be made to implement its software components. This section reviews the software developed in the literature, explains the alternative options, and justifies the implementation choices made for the developed AIP. Similarly to the hardware choices explained in the previous section (Section 4.3), the used software development tools have an impact on the final capabilities of the AIP. However, the made choices should also be compatible with all the low-level and high-level objectives of the AIP. Moreover, the software choices need to be usable on and with the chosen hardware components. Fortunately, this compatibility problem was already dealt with in during the hardware review.

The next paragraphs are each focused on one software dilemma and each point is reviewed in three steps. First, some theory is given about the concerned field of interest, then a overview of the relevant literature is performed, and finally the choice made in the context of the development of the software side of the AIP is justified. The topics that are analyzed hereafter are: the programming framework, the networking protocol, and the DataBase (DB) technology alongside the data formats.

Firstly, concerning the electronic programming framework choice, the most important decision to make is the programming language choice. The only real restriction that is problematic is the programming language compatibility with the hardware. Regarding the chosen computing unit, the RPI Zero 2W is compatible with all popular programming languages (and almost any existing language). That is one strength of this platform. However, concerning the used sensors and actuators, it is probable that their vendor or producer libraries are only available for the Arduino platform and eventually for the Raspberry Pi one.

In the literature, the used programming languages vary according to the project objectives. In the articles [Van et al., 2019] and [Park and Kim, 2021] the Python and Arduino C languages are used. In the articles [Jaenuri et al., 2021] and [Lin et al., 2017] only the Python language is used. And in the articles [Chowdhury et al., 2020], [Debauche et al., 2020], [Abdelouahid et al., 2020a], and [Eduardo et al., 2017] only the Arduino C language is used. As a side note, some articles standout: a non specified programming language was used in the articles [Chowdhury et al., 2020] and [Lin et al., 2017] to develop a graphical interface; the article [Lukito and Lukito, 2019] uses the PHP, Java Script, and HTML Script languages; the programming language used for the ML development in [Debauche et al., 2020] is not specified; in [Eduardo et al., 2017], some non specified programming languages are used to develop the Flask REST API and the graphical interface.

Fortunately, the devices Arduino libraries could be used on the RPI hardware thanks to a dedicated framework (i.e., RasPiArduino[5]). However, this choice is not interesting because the use of such a framework only gathers the downside of each platform without having their respective advantages. To get the most out of the chosen hardware, the Python language is the best option: it is very powerful, has a huge community support, and is very popular for electronic components interfacing. These reasons explain why this programming language was always used in the literature when the project is RPI based. When using Python, no specific programming framework is required to be used in the context of this master thesis. The electronic programming language chosen for the development of the software side of the AIP is thus Python. This choice can be observed in Figure 4.6.

Secondly, regarding the programming framework for the Management Dashboard, multiple choices have to be made: the frontend and backend frameworks. Fortunately, these two decisions can be taken completely independently. For the frontend part, two frameworks are currently the leaders: Flutter and Ionic React. Both provide the required tools to build native, cross-platform, responsive, and dynamic applications. The main high-level differences between the two are their release date (i.e., Flutter is more recent that Ionic React) and their logic (i.e., Flutter is an all in one package whereas Ionic React uses multiple subsystems to build the native application). For the backend part, almost any programming language could work because various frameworks are available. Among the most popular ones: Python with Django[6] or Flask[7]; JavaScript with Node.js[8], AngularJS[9], Vue.js[10], or React[11]; Go with Gin[12] or Beego[13].

From the reviewed articles, almost none mentions anything about the backend or frontend part of the user graphical interface. Three exceptions can still be mentioned: the article [Lukito and Lukito, 2019] uses the PHP, Java Script, and HTML Script languages for entirety of its developments (thus including the user interface) and the two articles [Abdelouahid et al., 2020a] and [Abdelouahid et al., 2020b] mention the use of dashboards from the smart home field (i.e., OpenHAB and Home Assistant).

For this AIP development project, the chosen frameworks are Flutter and Django. Both were chosen for their actual and efficient approach when it comes to build web and mobile applications and for their compatibility with later programming choices (e.g., the chosen database technology). As it is clarified later in this work, no full fledged backend was implemented but a complete frontend was developed. These choices can be seen in Figure 4.6.

Thirdly, for the networking protocol decision, the choice is more crucial. This is because of two

[5]RasPiArduino repository: `https://github.com/me-no-dev/RasPiArduino`
[6]Django web page: `https://www.djangoproject.com/`
[7]Flask web page: `https://flask.palletsprojects.com/en/2.1.x/`
[8]Node.js web page: `https://nodejs.org/en/`
[9]AngularJS web page: `https://angularjs.org/`
[10]Vue.js web page: `https://vuejs.org/`
[11]React web page: `https://fr.reactjs.org/`
[12]Gin web page: `https://gin-gonic.com/`
[13]Beego web page: `https://beego.vip/`

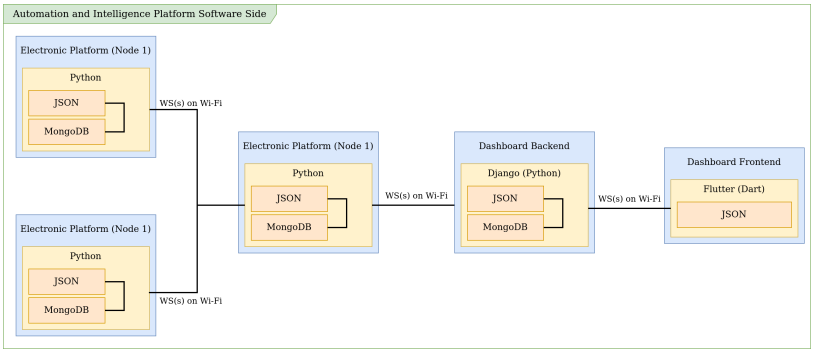

Figure 4.6: The software side of the AIP.

particularities of the AIP implementation project: bidirectional communications and automation in the server node. For the first one, it is in fact needed that the nodes that composed the software side of the AIP can exchange messages and information to all other nodes. This thus means that multiple one-way links need to be used or that a single two-way communication system is needed. For the second particularity, the automation of the server, it means that it could be useful to have the control over the communication server as well as on the clients. If such a feature is possible, some automation or cognitive computing algorithms could be used within the server itself to monitor and improve the AIP. In addition to these two particularities, multiple choices also need to be made. Because the current project controls both the hardware and software side of the AIP, a network protocol and a network communication protocol need to be chosen.

From the literature, almost nothing is said about the used network communication protocols. Two main explanations are that either the developed system is single-node or the used network protocol also embeds a default communication protocol (e.g., Zigbee, Z-Wave, etc.). The only exceptions are the two articles [Abdelouahid et al., 2020a] and [Abdelouahid et al., 2020b] that use the MQTT communication protocol. Thankfully, more details are sometimes given about the used networking protocol. The articles [Chowdhury et al., 2020], [Lukito and Lukito, 2019], [Debauche et al., 2020], [Abdelouahid et al., 2020a], [Eduardo et al., 2017], [Cabaccan et al., 2017], and [Abdelouahid et al., 2020b] use Wi-Fi. The article [Lin et al., 2017] mentions the use of Bluetooth while [Debauche et al., 2022] is specifically designed around the 5G LTE network. Finally, the Zigbee protocol is also mentioned in the literature review of the articles [Van et al., 2019], [Park and Kim, 2021] and [Cambra et al., 2018].

The main trend deduced from the literature is that the dominant networking protocol is Wi-Fi and that the only mentioned communication protocol is MQTT. For this master thesis, the networking protocol choice is fairly easy. Seeing that the developments are made in a restraint environment, the use of the 4G or 5G LTE network is non needed. Moreover, the Zigbee and Z-Wave protocols are both low-energy low-data-rate and this project could require a large bandwidth to transfer images or point-cloud data. The best choice is thus the use of the Wi-Fi networking protocol. Unfortunately, it does not include a default bidirectional communication protocol, a specific one thus has to be chosen. Multiple choices are available but only these three communication protocols are interesting for this project: HTTP(s), MQTT, and WS(s) (Websockets). The MQTT choice is the easy choice, it would work perfectly for the current needs. However, no control over the server (MQTT broker) automation is possible. The HTTP(s) choice allows the customization of the server automation but was designed for one-way communications. It is true that it is possible to perform bidirectional data exchanges over HTTP(s) but the last choice is better. The WS(s) network protocol is the best choice here. In fact, it uses websockets to create a bidirectional communication channel between two clients. Thanks to this functioning, no server is needed (excepted for the connection opening that uses the HTTP(s) handshake system) and the automation of all clients can be customized. The final choices for the network stack are thus the WS(s) communication protocol

over the Wi-Fi networking protocol. These choices can be viewed in Figure 4.6.

Fourthly, regarding the DB technology alongside the data formats choices, two main choices need to be made: the data format used throughout the whole project (from the electronic device to the user preferences) and the data storage system (e.g., database, data lake, etc.).

In the already analyzed articles, absolutely nothing is said about the used data format. In fact, no other project needed to care about it because none of them communicates over websockets. Concerning the data storage system, some information is given in scientific papers reviewing system architectures and cloud infrastructures but they are not of interest here because this AIP development project does not use these predefined architectures and infrastructures.

The choice of the data format is pretty easy. When looking at the requirements of the system, it can be seen that all information exchange within the project required some information about the message origin, destination, content, etc. This is because the use of the WS network communication protocol forces the development of a message routing system to transmit messages from node to node. The obvious choice is thus the use of the JSON data format because it is structured, typed, and integrates well with the other software choices. Concerning the choice of the data storage system, three main concepts are available: DataBases, data lakes (or data oceans), and data warehouses. The DB is a collection of data that allows the system to store, analyze, and interact with the data. Two types of DB exist (i.e., relational and non relational) but both can be structured or semi-structured according to the stored data. The data warehouse is an advanced DB system that stores highly structured information from various disparate sources and that is optimized for analytic operations. The data lake is a data warehouse that stores data in its original raw format. To choose among these three systems, the best thing to do is to analyze the developed AIP. In this case, seeing that the data source are gathered together under the AIP node system and seeing that the used data format is standardized for the websockets, the best choice is to use a non-relational database such as MongoDB[14]. Moreover, this choice enables the use of the same data storage system on all nodes of the developed AIP and that enforces the AIP robustness by allowing to temporarily store data in the local node if some networking outage occurs. The final choices are thus to use the JSON data format with a MongoDB database for each and every node of the AIP. These decisions can be seen in Figure 4.6.

To conclude this section about software choices, an additional fact should be explained. As for the hardware decisions, no currently popular system could have been a wrong choice. The decisions taken in this section are obviously the result of deep research and are thus the best possible choices for the current development but choosing the HTTP(s) protocol instead of the WS(s) one will not have broken the system. This is because most of the currently available platforms can be hacked to make them work as expected even if they were not designed to perform some operations. The only real still relevant constraint when choosing software components is the hardware compatibility. The convenience of the wide compatibility of the RPI platform is also one of the reasons why it was chosen. Finally, the remark made in the hardware review section (Section 4.3) about the homogeneity or heterogeneity and general purpose or specialized dilemmas is still valid here because the developed AIP is intended to evolve and is opened to improvements. The emphasis was thus put on the homogeneity and general purpose of the software developments.

[14]MongoDB web site: `https://www.mongodb.com/`

Chapter 5

Hardware developments

Thanks to the theoretical background reminders made within Chapter 2 and the methodology explanations supplemented with the technology choices justifications from Chapters 3 and 4, it is now possible to describe the hardware systems that were developed. In this section, hardware assemblies are reviewed in details. That means that the built Proof Of Concept (POC) is detailed and that a more advanced design is shown. This additional hardware system was studied but not built during the development of the hardware side of the AIP.

Before reviewing the done hardware developments, it is important to make a comment. As stated in the introduction (Chapter 1), this master thesis is focused on the ICT side of the SPFs. This focus has an impact on the developed electronic systems. In fact, seeing that the main goal of SPFs is to produce plants by providing them the best environment possible, the electronic systems should be useful to reach that goal. However, because of the focus of this project, the relevance of each hardware component is not deeply studied. The methodology used to develop useful electronic systems thus follows the developments studied in the literature review (Section 2.5).

5.1 Current Proof Of Concept

To implement the hardware side of the AIP, a POC of the electronic system was built. This assembly has the objective of fulfilling all the basic requirements of the AIP. Moreover, being the first developed system, this POC also has the objective of testing the made hardware choices. Figure 5.1 presents the electrical wiring diagram of this first electronic assembly and Figure 5.2 shows the connection diagram of the same assembly. As it can be seen on these schematics, two Raspberry Pis (RPIs) share the sensors and actuators. This is is not because of a computing power limitation or a lack General Purpose Input/Output (GPIO) pins, it is because of the physical distance between the various devices. In fact, the water probes and the relays are located near the water reservoir while the other sensors are near the cultivation tray. This observation is important because it means that the hardware need to be modular. Moreover, as it will be seen in the software review (Section 6), this modularity should be (and is) present in the software side of the AIP.

The pictures and references of the used hardware components are presented in Appendix C but here is a quick view of the system. The used Single Board Computers (SBCs) are Raspberry Pi Zero 2W. The first one is linked to 2 humidity and temperature sensors, 2 waterproof temperature sensors, and a multiple sensors Hardware Attached on Top (HAT) which has a temperature and humidity sensor as well as a light intensity detector and a barometric pressure sensor. The second one is used to control a 4 relays HAT and an 8 channels Analog to Digital Converter (ADC) HAT. The 220v-AC lighting system and the nutrient solution pump are connected through the 4 relays HAT and the 3 analog water quality probes are interfaced via the 8 channels ADC HAT.

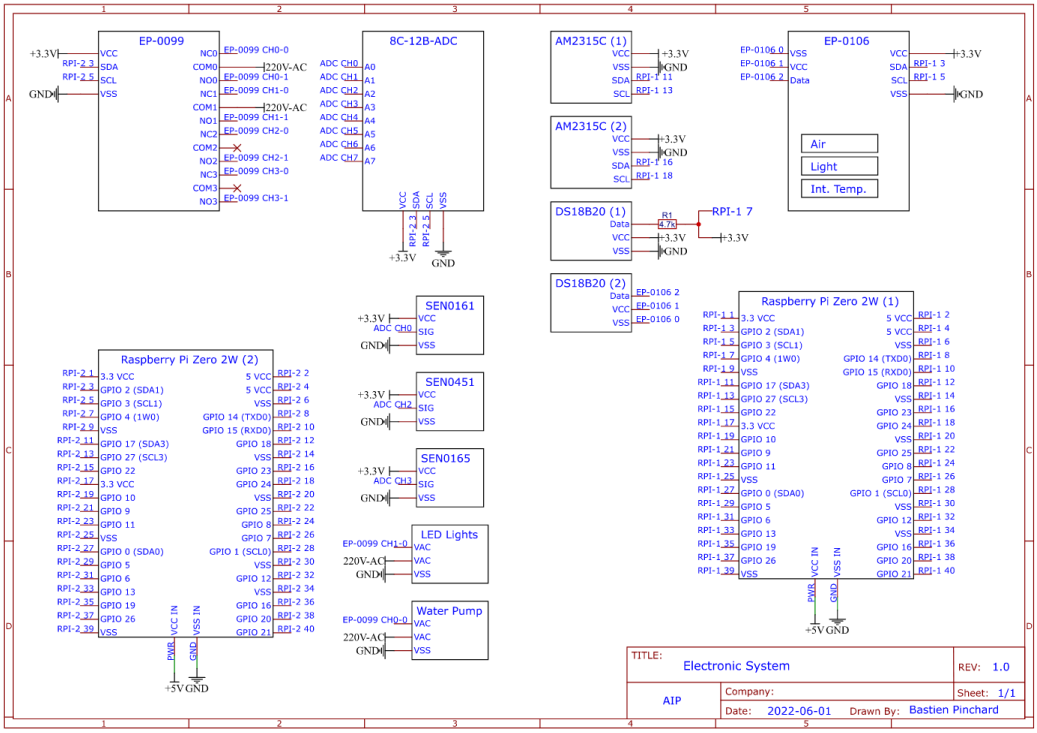

Figure 5.1: The electrical wiring diagram of the hardware POC assembly implementing the electronic system of the AIP (AIP hardware side).

The humidity and temperature sensors (AM2315C, Figure C.1) are digital and use the I2C communication protocol. Their default address is 0x38 and it cannot be changed, this means that multiple I²C channels or an I²C multiplexer have to be used.

The waterproof temperature sensors (DS18B20, Figure C.2) are digital and use the 1-wire communication protocol. A 4.7K pull-up resistor has to be used but multiple of these sensors can be chained together and connected to the same I/O pin.

The multiple sensors HAT (EP-0106, Figure C.3) uses the digital I²C communication protocol with the non-changeable default address 0x17. The useful sensors present on this board are: onboard temperature and humidity, barometer (with temperature and humidity), light level, and external waterproof temperature.

The 4 relays HAT (EP-0099, Figure C.4) also uses the digital I²C communication protocol with the default address range from 0x10 to 0x13. This means that 4 HATs can be stacked to control up to 16 relays. This system is very handy on the programming side but not very durable on the electronic side. This is because the screws are not accessible when the HAT is stacked with other hardware.

The 8 channels ADC HAT (8C-12B-ADC, Figure C.5) also uses the digital I²C communication protocol with the non-changeable default address 0x04. It has a 12-bits precision, an internal real-time clock, and it can be used in comparison mode to compare the results of two channels.

The pH probe (SEN0161, Figure C.7) is an analog probe that includes a daughter board that transforms the probe measurement into an analog signal. This water quality probe quantifies the Potential of Hydrogen (pH) of the solution it is immersed in. This probe needs to be calibrated

before using it and its measurements can drift overtime.

The EC probe (SEN0451, Figure C.8) also is an analog probe that includes a similar daughter board with the same functionality as the pH probe daughter board. This water quality probe quantifies the Electrical Conductivity (EC) of the solution it is immersed in. This probe also needs to be calibrated before using it and its measurements can thus drift overtime.

The ORP probe (SEN0165, Figure C.9) is similar to the two previous probes: it is analog and comes with a similar daughter board. Moreover, it also needs to be calibrated and its measurements can drift overtime. This water quality probe measures the Oxidation-Reduction Potential (ORP) of the solution it is immersed in.

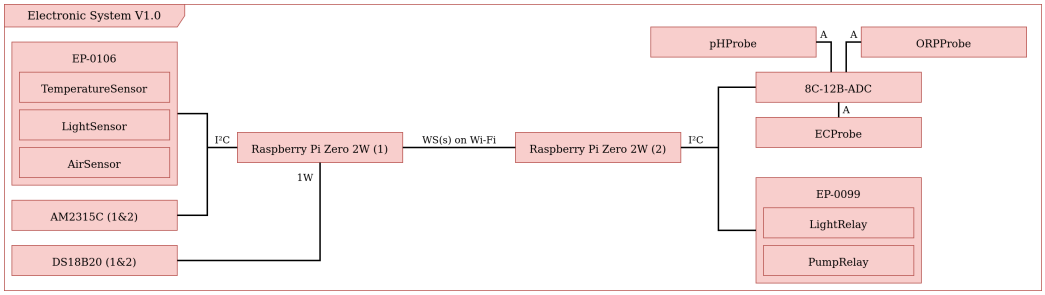

Figure 5.2: The connection diagram of the hardware POC assembly implementing the electronic system of the AIP (AIP hardware side). The used communication protocol is shown for each link: A (Analog), D (Digital), I²C, PWM, or 1W.

The objectives of the electronic systems that compose the hardware side of the AIP are to remove as much as possible human-physical interactions with the CSMs and the SPFs. To do so, the electronic system has the role of collecting data about the environment to be able to verify its optimality and modify its state. For this reason, the most modular the hardware system is the better it is because new measurements and controls will always be implemented into the AIP. The current set-up is not fully featured but is modular. The already implemented control systems are: the water flow management, the lighting activity (on/off), the water quality measurements, and the environment status. The next objectives for the control system are: the lighting management (on/off, dimming level, and Kelvin temperature), the water quality management (measurements and modifications), and the environment management (measurements and modifications). Another downside of the current POC is that HATs were used. Even if they are really helpful for the programming and crafting sides of the project realization, they are not the ideal components because they lack some important modularity. In fact, the used HATs are very flexible (which is important when implementing a first POC) but are not modular. For example, the ADC comes with an embedded microcontroller (i.e., STM32F030F4P6TR or MM32F031F6P6). The presence of this microcontroller makes the HAT really flexible by making its interfacing accessible but its computing capabilities are useless for this project seeing that the use of microprocessors was preferred. Moreover, the same HAT is not modular because multiple ADCs cannot be used in the same system (because the assembly form factor does not allow the use of an I²C multiplexer and because the ADC I²C addresses are non modifiable). The reality is that, as for all the other sensors, actuators, and probes, these components were chosen to develop the hardware assembly POC because of their accessibility (regarding the price, the required soldering, etc.) and their flexibility. This is the reason why some advanced designs are shown hereafter. However, even with these downsides, these hardware components are perfect fit for developing the current POC that supports the hardware side of the AIP.

5.2 Advanced design

To overcome the limitations of the build hardware assembly POC, a more advanced concept of the control system was designed. As said earlier, the objectives this better control system should fulfill are: the lighting management (dimming level, and Kelvin temperature), the water quality management (measurements and modifications), and the environment management (measurements and modifications). Obviously, the monitoring systems featured in the previous attempt should be kept. These monitoring systems are: the water flow management (on/off), the lighting activity (on/off), the water quality measurements (e.g., pH, EC, ORP, etc.), and the environment status (e.g., air temperature and humidity, etc.). Moreover, this advanced concept should remove the modularity limitations of the first POC by not using HATs with redundant microcontrollers and by designing a modular assembly.

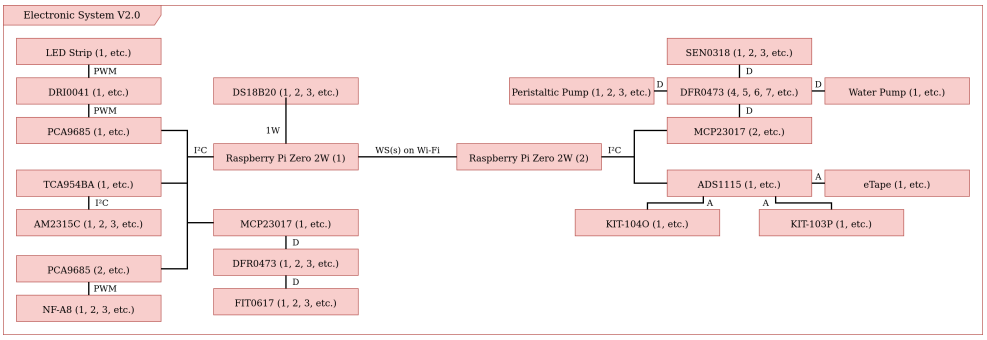

Figure 5.3: The connection diagram of the advanced hardware assembly for future version of the electronic system of the AIP (AIP hardware side). The used communication protocol is shown for each link: A (Analog), D (Digital), I²C, PWM, or 1W.

Figures 5.4, 5.5, and 5.6 present the wiring diagrams of this new electronic system while Figure 5.3 shows the connection diagram of the same assembly. As it can be seen in these figures, the main focus was to emphasize the hardware expandability and modularity. In fact, in addition to the fact that many more electronic components are used in this second version, less RPI GPIO pins are needed and no I²C address conflicts are present (thus removing the need of the use of multiple I²C buses). These particularities and the implemented features are reviewed in the next paragraphs. Additionally, the used electronic components are shown in Appendix C.

Concerning the first page of this new electronic system wiring diagram (Figure 5.4), three main improvements can be found in addition to the RPI already used in the first system. Firstly, the waterproof temperature sensors (DS18B20, Figure C.2) are still present but chained together thanks to their use of the 1W bus. Similarly, the temperature and humidity sensors (AM2315C, Figure C.1) are also still present but are connected to an I²C multiplexer (TCA9548A, Figure C.14) that allows the usage of multiple devices with the same address. Then, the third improvement concerns the lighting system. This new electronic system uses a PWM driver (PCA9685, Figure C.11) and a DC motor driver (DRI0041, Figure C.12) to drive and modulate some tuneable LEDs strips (Figure C.13). This lighting equipment is capable of controlling the light intensity thanks to the use of PWM and the light temperature thanks to the two channel design of the LEDs strip.

On the second page of the new electronic system wiring diagram (Figure 5.5), two particularities are notable. Firstly, the reuse of a PWM driver (PCA9685, Figure C.11) coupled with fans enables the system to control a bit more its environment. One could imagine to use this system within an enclosed cultivation chamber to start better controlling the culture environment. Then, the second particularity is about the water-flow management. The addition of solenoid valves (FIT0617, Figure C.18) controlled with power relays (DFR0473, Figure C.16) and a GPIO

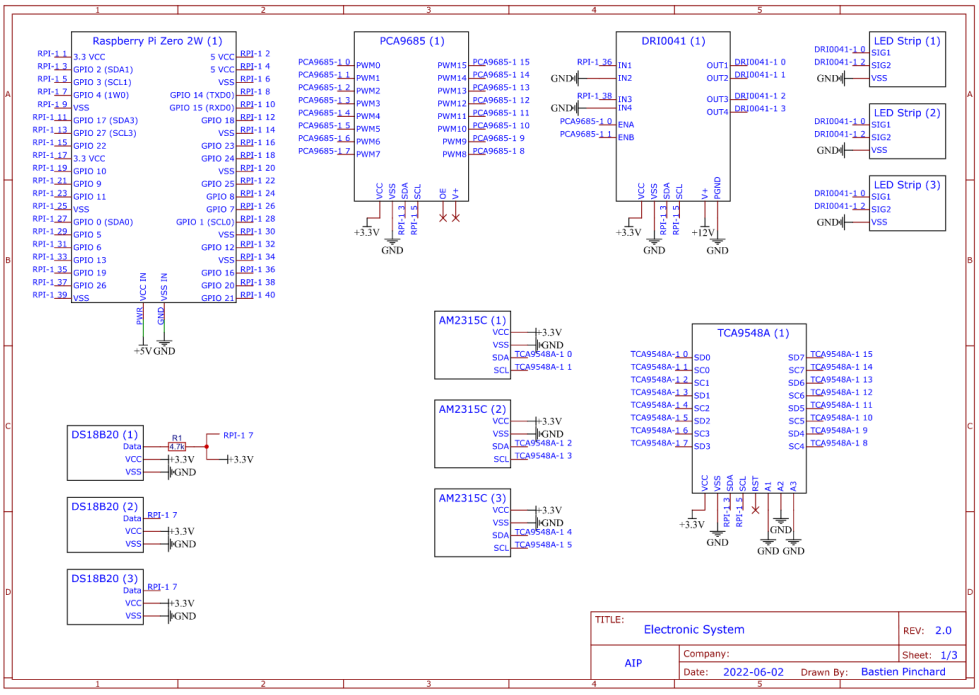

Figure 5.4: The first page of the electrical wiring diagram of the advanced hardware assembly for future version of the electronic system of the AIP (AIP hardware side).

I²C expander (MCP23017, Figure C.15) makes this system more modular regarding the nutrient solution distribution.

Figure 5.6 presents the third and last page of the wiring diagram of the new electronic system. This last part contains three interesting points. In addition to the use of a second RPI as in the original design, the reuse of a GPIO I²C expander (MCP23017, Figure C.15) to control power relays (DFR0473, Figure C.16) that themselves control peristaltic pumps (Figure C.19) and a water pump enables this new electronic system to dispense some nutrient mixes to adjust the nutrient solution. Moreover, the use of external liquid level sensor (SEN0368, Figure C.20) makes the control of the nutrient mixes stock tractable. Thirdly, the last interesting point of this wiring diagram page is the inclusion of an I²C ADC (ADS1115, Figure C.21) to control the measurements made by the water quality probes (KIT-103P, Figure C.23 and KIT-104O, Figure C.24) and the immersed liquid level sensor (Figure C.22).

To end this review of the advanced electronic design, two main points are noteworthy. The first one concerns the expandability of the system. As it can be seen in the Figures 5.4, 5.5, and 5.6, each component is labeled with an index in parentheses. This is not insignificant because it means the use of multiple components for each device was studied. Thanks to the fact that all "controller boards" (i.e., the I²C multiplexer, the PWM driver, the GPIO I²C expander, and the I²C ADC) use the I²C protocol and have a settable address, each of them can be chained together and used at the same time. This is an important feature because it allows the currently developed systems to be expanded. The second interesting aspect of this advanced electronic design is about its modularity. In fact, thanks to its expandability, many GPIO pins are still available to add new functionalities to the control system. Moreover, all sub-systems (e.g., the lighting control) are independent from each other. It is thus possible to compose the perfect system for each culture chamber (e.g., it is possible to add the temperature and humidity module of the first RPI from the first page of the wiring diagram to the second RPI from the third page of the same schema). Finally, as a bonus point, if some lack of GPIO pins is encountered, the use of a GPIO I²C expander is possible (as it

was done for the power relay control). Moreover, to interface some 5V devices with the 3.3V RPI, an I²C safe bidirectional logic level converter (BSS138, Figure C.25) can be used.

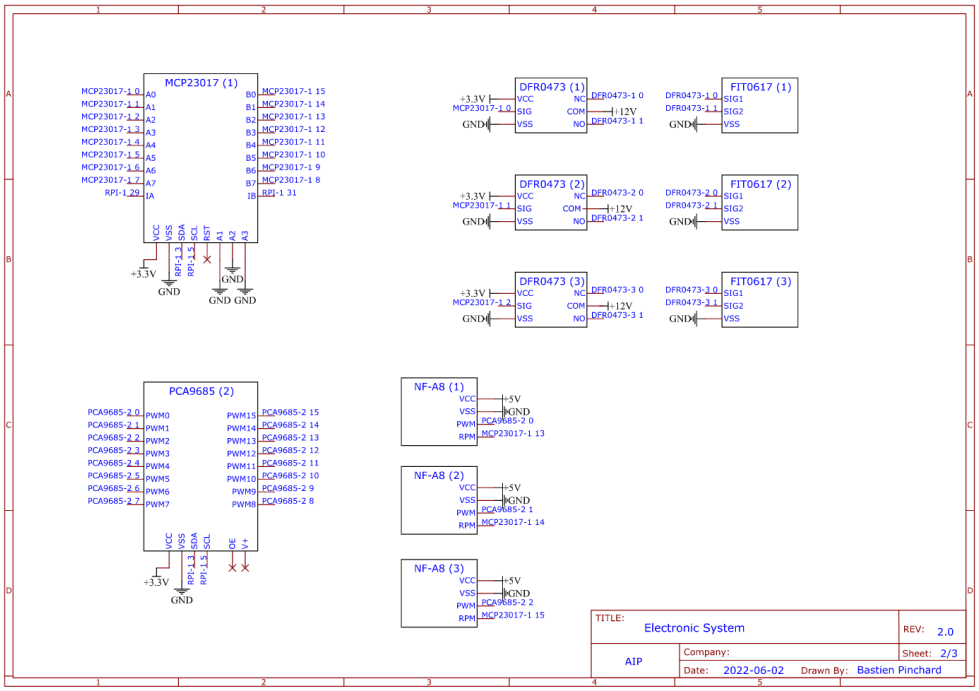

Figure 5.5: The second page of the electrical wiring diagram of the advanced hardware assembly for future version of the electronic system of the AIP (AIP hardware side).

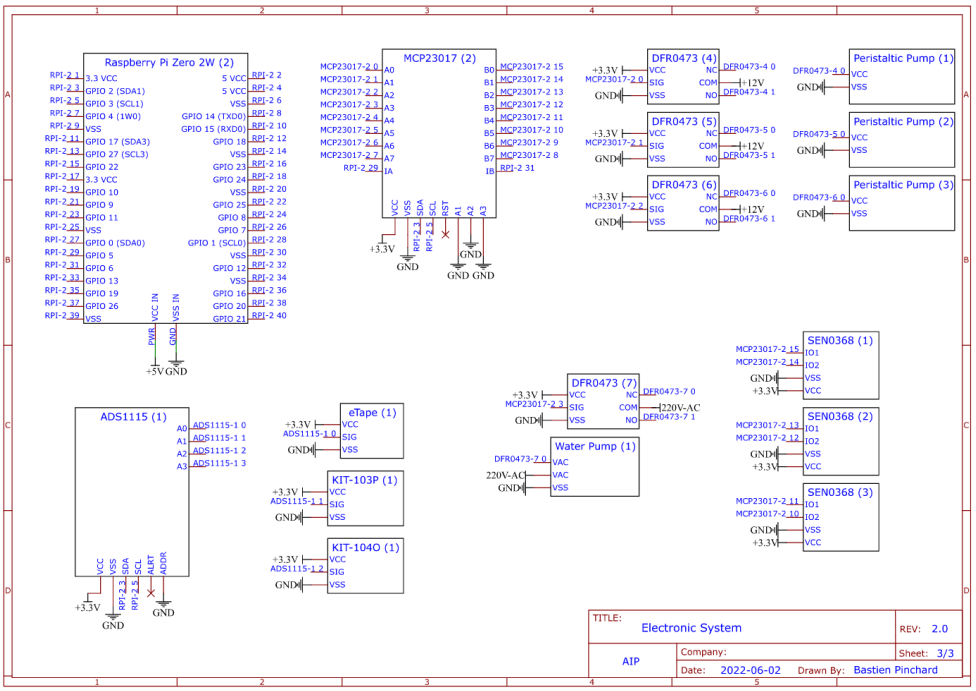

Figure 5.6: The third page of the electrical wiring diagram of the advanced hardware assembly for future version of the electronic system of the AIP (AIP hardware side).

Chapter 6

Software developments

The Electronic Management System (EMS) and the Management Dashboard (MD) are the two complementary software components that implement the software side of the AIP. This section is devoted to review the Minimum Viable Product (MVP) of the control software. Both sides of this MVP are detailed in terms of high-level organization and low-level implementation. Even if an AIP has to have a human management system (here, the MD), the EMS is the most original piece of software in this work. In fact, the EMS contains the core concepts of the AIP (e.g., the environment sensing, the automation algorithms, etc.). Figure 6.1 shows the characteristics and the interactions between these software components.

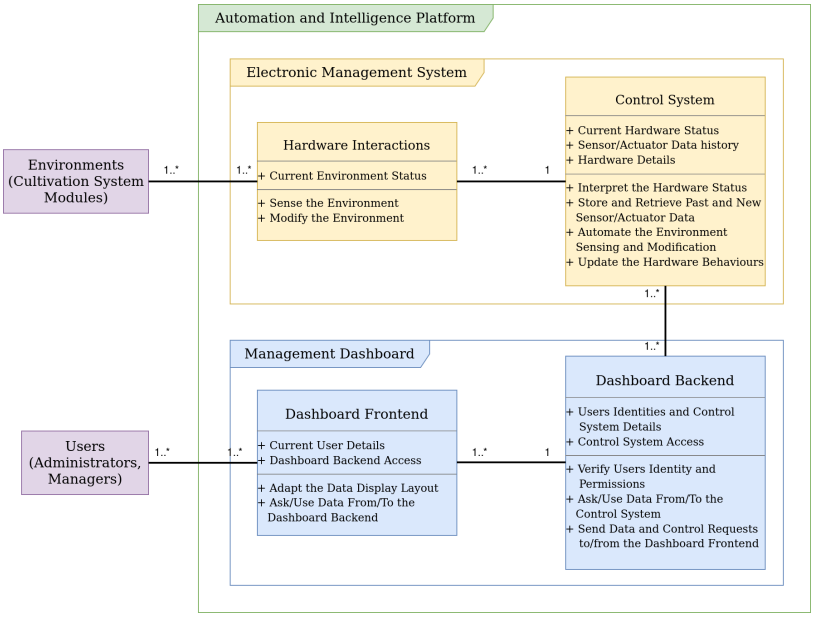

Figure 6.1: High level component diagram of the AIP software side. The EMS and its components are color-coded in yellow, the MD and its frontend/backend parts are represented in blue, the sum of these two systems composed the AIP colored in green, and the external elements are in purple.

One interesting feature of the software side of this AIP is that it was built to support a variable number of CSMs and multiple users (i.e., the people who operate the Smart Plant Factory). Moreover, multiple SPFs could be run in parallel and one user could have access to each SPF on the MD. Additionally, the control system and the dashboard backend can respectively serve multiple hardware components and dashboard frontend. This was made possible thanks to modularity

provided by the object oriented approach used both for the EMS and the MD.

6.1 Electronic management system

The Electronic Management System (EMS) is composed of two main components: the control system and the hardware interactions software. Before describing each implementation, the major technical details can be listed. As a side note, it is worth knowing that these are based on the technical choices described in Sections 4.3 and 4.4. Moreover, additional details (i.e., line counts, external authors copyright, etc.) about these source code and a repository link can be found in Appendix A:

- *Python* 3.8 or above.
- The *abc* standard module for abstract classes.
- The *argparse* and *configparser* modules for command line arguments parsing and configuration file analysis.
- A unique MongoDB database (*pymongo* module) with a collection for each implemented device and the *json* module to efficiently parse communications.
- The *asyncio* module to make the execution asynchronous.
- The *websockets* module to handle bidirectional communications.

In addition to these choices, two programming concepts were used to improve the system modularity and to support the object oriented approach. These two programming paradigms are the abstraction and the inheritance. For the first one, abstraction, it is defined as "the creation of abstract concept-objects by mirroring common features or attributes of various non-abstract objects or systems of study" [Kramer, 2007]. For the second one, inheritance, it is described as "deriving new classes (sub classes) from existing ones such as super class or base class and then forming them into a hierarchy of classes" [Ralph and Brian, 1991]. In this project, the two concepts were used together to create classes (i.e., the hardware interactions source code) that are built upon one base class (i.e., the control system source code) to specify new functionalities while maintaining the same basic behaviors.

As it was discussed in Sections 4.3 and 4.4, the AIP automation algorithms and cognitive computing systems can come from two different sources: the cloud or the edge computing. In this project, this is the EMS that is responsible of connecting the electronic component together and to the cloud (if any). Thanks to this custom EMS, the AIP developed during this master thesis is compatible with the two paradigms: it can reach the cloud to ask what actions to perform and it can compute its next action independently thanks to the edge computing. As it is shown hereafter, the control system is responsible of both paradigms while the hardware interactions source code are only concerned about the edge computing.

6.1.1 Control system

The control system is the most interesting software piece here because it was designed from the ground up to support the AIP. As it can be seen in Figure 6.2, the system is composed of seven abstract classes with a precise inheritance tree. This specific structure makes the source code more compact, reusable, and flexible. Each class is responsible of given tasks and the hardware interactions implementations are vastly improved and easier to develop thanks to the lowest level abstract classes (i.e., VSensor and VActuator). Additionally, four out of these seven abstract classes are in control of the inter-component bidirectional communications. These can be seen in the orange boxes in the figure 6.2. The network communications are an important part of the EMS seeing that it allows the usage of multiple hardware systems on which the software components can

be deployed. Furthermore, each one of these four components represents a different control level in which it is possible to analyze, aggregate, verify, and transmit information. This structure thus makes the EMS very modular and allows a fine grained control over the hardware components. Finally, when looking at Figures 6.2 and 4.3, it can be observed that the four communications responsible classes follow the exact same structure as the four main components of SPFs. This similarity is wanted to ensure the homogeneity of the system and it shows that the maximum controllability level is available with this EMS structure.

These four classes (i.e., VHuman, VFarm, VCsm, and VDevice) are not the only available levels for implementing decision making. In fact, the common base class (i.e., VClass) could also be used to implement a global decision making system and the two children classes (i.e., VSensor and VActuator) do also have the same capabilities. This local computing power is often called edge computing. However, as mentioned above, these classes are also compatible with the cloud computing paradigm. In fact, thanks to the VClass class, an HTTP connection towards a cloud could easily be setup and shared to each device based on the children classes.

As a side note, the naming scheme used in the EMS is pretty simple: each class name is prefixed with a "V" (which stands for "Virtual") to refer to the fact that they are abstract classes and the suffix of the class names refer to the component they serve (e.g., the VFarm class is the abstract class made for the SPF control object.).

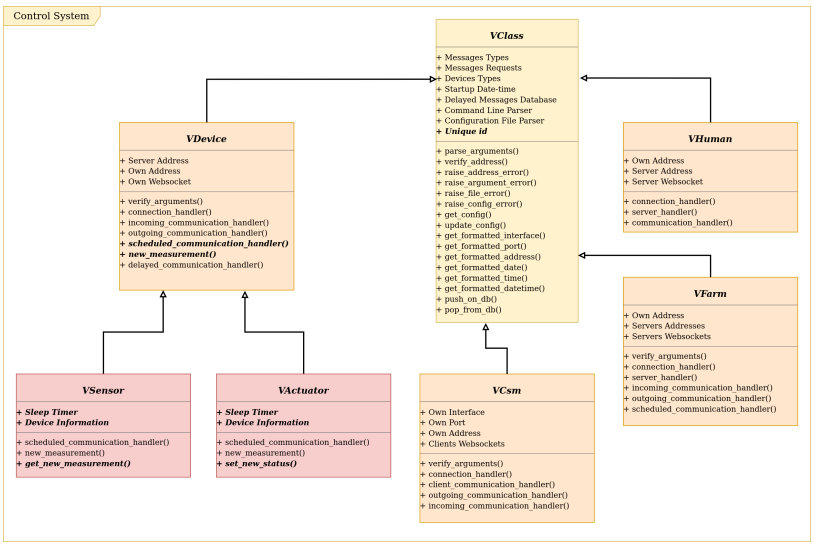

Figure 6.2: Class diagram of the electronic control system that composed the EMS. The inheritance levels are color-coded from yellow to red (from pure software to software that interact with hardware) and the abstract components are written in bold italic.

The abstract class VClass is the base class for each and every device of the EMS. This class is an abstract class because its direct children classes are abstract (i.e., VDevice, VCsm, VFarm, VHuman) and because it requires them to implement the device identification field. The VClass class implements the command line parser, the configuration file parser, a temporary MongoDB database, and some utility functions. All these functions provide a common interface for each of the device class. This standardization ensures that every system is built with the same foundations and that they are all intercompatible. For example, the definition of the command line parser and of the configuration file parser provide two important functionalities to every sub-classes. Moreover, the definition of the temporary database and of its push/pop functions ensures that every sub-class can have access to such a database in case of network outage (i.e., to store messages and send them as soon as the network connection is established). A specificity of this VClass class is that the

class variables are used to define macros for network communications. These macros are used by all children classes and allow an efficient parsing of websocket messages and tied up the code base by strengthening the logic guard rail. These macros concern the message type and requests but also define the available device types. All the aforementioned details can be observed in the pseudo code provided in Listing 6.1.

```
# Libraries includes
import pymongo # For the database
import datetime # For time formatting
import argparse # For command line arguments
import configparser # For the configuration file

# Classes includes
from abc import ABC # For abstract class error management

class VClass(ABC):
    def __init__(self) -> None:
        # Set-up the basic class variables
        self.startup_datetime = None
        self.parser = None
        self.config = None
        self.delayed_db = None
        # Define the message types
        self.GET = 'GET'
        self.POST = 'POST'
        # Define the message requests
        self.NEW_MEASUREMENT = 'NEW MEASUREMENT'
        self.SCHEDULED_MEASUREMENT = 'SCHEDULED MEASUREMENT'
        self.DEVICE_INFO = 'DEVICE INFO'
        self.DEVICE_CONFIGURATION = 'DEVICE CONFIGURATION'
        self.DEVICE_CALIBRATION = 'DEVICE CALIBRATION'
        # Define the device types
        self.SENSOR = 'SENSOR'
        self.ACTUATOR = 'ACTUATOR'
        self.FARM = 'FARM'
        self.CSM = 'CSM'
        self.HUMAN = 'HUMAN'

    def __call__(self) -> None:
        self.parse_arguments()
        self.startup_datetime = self.set_datetime()

    def parse_arguments(self) -> None:
        # Define the command line parser
    # Verify the user-given addresses

    def verify_address(self, server_address) -> bool:
        # Verify if the address match a specific pattern

    def raise_errors(self, **kwargs) -> None:
        # Raise the wanted error for the parser, config-parser, address or file

    def get_config(self, file_path) -> dict:
        # Parse, evaluate, and return a configuration file

    def update_config(self, file_path, fields) -> None:
        # Update a configuration file

    def get_formatted_interface(self, address) -> str:
        # Format and return the interface number

    def get_formatted_port(self, address) -> str:
        # Format and return the port number

    def get_formatted_address(self, interface, port) -> str:
        # Format and return the websocket address

    def get_formatted_date(self) -> str:
        # Format and return the current date

    def get_formatted_time(self) -> str:
        # Format and return the current time

    def get_formatted_datetime(self) -> str:
        # Format and return the date and time

    def push_on_db(self, message) -> pymongo:
        # Push a new record on the object database (create the database if necessary)

    def pop_from_db(self) -> dict:
        # Pop the first message pushed on the database (signal when the database is/becomes empty)
```

Listing 6.1: VClass abstract class pseudo code

The VDevice class directly inherits from the VClass abstract class. This VDevice class is an abstract class too because it defines two abstract methods that should be implemented in the direct children classes (VSensor, VActuators, etc.). These two methods provide a common interface to ask a new measurement and handle scheduled communications. The first one is needed to standardize the action of getting a new measurement because sensors and actuators have opposite functioning: actuators cannot take measurements but can modify their state and sensors cannot change their state but can measure their environment. The second abstract method is also needed to ensure that each children class can define its own schedule to send new data and what data should be sent. The

VDevice class also implements the websocket management and communication handlers thanks to asynchronous primitives (i.e., async and await). For the networking component, the VDevice class is a client that is able to send and receive messages by connecting to a server. Moreover, this client can deal with network errors and automatically reconnect to the server when available. These communications are managed asynchronously and within different task handlers. All these particularities can be observed in the pseudo code provided in Listing 6.2.

```
# Libraries includes
import abc # For abstract methods
import asyncio # For asynchronous communication handling
import websockets # For bidirectional communications

# Custom classes includes
from vclass import VClass # For inheritance

class VDevice(VClass):
    def __init__(self) -> None:
        super().__init__()
        # Define the network addresses
        self.server_address = ''
        self.own_address = ''
        # Prepare the client web-socket creation
        self.own_websocket = None

    def __call__(self) -> None:
        try:
            # Launch the infinite loop
            super().__call__()
            self.verify_arguments()
            asyncio.run(self.connection_handler())
        except KeyboardInterrupt:
            # Cleanly stop the infinite loop

    def verify_arguments(self) -> None:
        # Verify the provided command line arguments and raise errors if needed

    async def connection_handler(self) -> None:
        # Connect to the server (CSM) (with automatic reconnection)
        async for self.own_websocket in websockets.connect(self.server_address):
            # Launch the network listening and message sending
            task1 = asyncio.create_task(self.incoming_communication_handler())
            task2 = asyncio.create_task(self.scheduled_communication_handler())
            task3 = asyncio.create_task(self.delayed_communication_handler())
            tasks = [task1, task2, task3]
            try:
                await asyncio.gather(*tasks)
            except websockets.ConnectionClosed:
                # Clean the connection closing and try to reconnect

    async def incoming_communication_handler(self) -> None:
        async for message in self.own_websocket:
            # Analyze the incoming communications and execute the correct followup

    async def outgoing_communication_handler(self, message) -> None:
        if self.own_websocket:
            # Send the message to the correct destination
            await self.own_websocket.send(json.dumps(message))
        else:
            # Store the message if the connection is unavailable
            self.push_on_db(message)

    @abc.abstractmethod
    async def scheduled_communication_handler(self) -> None:
        # Abstract interface to periodically send messages

    @abc.abstractmethod
    async def new_measurement(self) -> json:
        # Abstract interface for asking a new measurement

    async def delayed_communication_handler(self) -> None:
        # Send the messages stored during the connection loss
        while message is not None:
            await self.outgoing_communication_handler(message)
            message = self.pop_from_db()
```

Listing 6.2: VDevice abstract class pseudo code

The VCsm class is the last abstract class that is used to implement the Cultivation System Module concept. This class handles the websocket management and manages the communications. As for the VDevice class, the communications are managed asynchronously and within different task handlers. Regarding the networking communications side, the VCsm class acts as a server that accepts connections and that routes communications if needed. The VCsm class is thus a bridge between VDevice and VFarm objects to allow them to cross communicate. Obviously, the VCsm class also has the ability to intercept messages that are intended for it. Moreover, this server manages its known clients and can deal with network outages. The client management system is made with a dictionary to keep track of the clients websockets alongside their information. With this dictionary, the server can analyze the clients information to know towards which client(s) messages should be sent. Thanks to these functionalities and thanks to the VClass base utilities, a basic CSM behavior

can be inherited from this class for any custom CSM implementations. As for all the abstract classes of the EMS, this VCsm class can be supplemented with automation algorithm and data analysis systems. Once again, Listing 6.3 provides a pseudo code in which all these details can be analyzed.

```
# Libraries includes
import asyncio # For asynchronous communication handling
import websockets # For bidirectional communications

# Custom classes includes
from vclass import VClass # For inheritance

class VCsm(VClass):
    def __init__(self) -> None:
        super().__init__()
        # Define the network addresses
        self.interface = ''
        self.port = ''
        self.own_address = ''
        # Prepare the clients connections
        self.clients_tuples = {} # [address]:(websocket, information)

    def __call__(self) -> None:
        try:
            # Launch the infinite loop
            super().__call__()
            self.verify_arguments()
            asyncio.run(self.connection_handler())
        except KeyboardInterrupt:
            # Cleanly stop the infinite loop

    def verify_arguments(self) -> None:
        # Verify the provided command line arguments and raise errors if needed

    async def connection_handler(self) -> None:
        # Serve the provided websocket address
        async with websockets.serve(self.client_communication_handler,
        self.interface, self.port):
        await asyncio.Future()

    async def client_communication_handler(self, client) -> None:
        # Route the messages to the wanted destination(s)
        async for message in client:
        if message['destination'] == self.own_address:
            await self.incoming_communication_handler(message)
        elif message['destination'] in self.clients_tuples:
            await self.outgoing_communication_handler(message)

    async def outgoing_communication_handler(self, message) -> None:
        # Send the message to the right client
        await self.clients_tuples[message['destination']][0].send(message)

    async def incoming_communication_handler(self, message) -> None:
        # Analyze the message type and request and launch the correct followup
```

Listing 6.3: VCsm abstract class pseudo code

The VFarm class also is the last abstract class that is used to implement the Smart Plant Factory concept. This class handles the websocket management and manages the communications. As for the previously presented abstract classes, the communications are managed asynchronously and within different task handlers. For the communications side, the VFarm class is a client that can connect to multiple servers (here, some VCsm objects) via the network at the same time to tie them all into one control system. Indeed, with this particularity, the VFarm objects can be used to transmit messages between systems. Moreover, this multi-servers client has the ability to manage its list of known servers and will automatically reconnect to them if network outages occur. The fact this client can be connected to multiple servers is not trivial. This particularity is implemented by creating a new websocket for each server and by keeping track of the currently available servers. Similarly to the VCsm class, the VFarm class is able to deal with network errors. Finally, as it can be seen in Listing 6.4, a basic SPF behavior can be inherited from this class for any custom SPF implementation.

```
# Libraries includes
import asyncio # For asynchronous communication handling
import websockets # For bidirectional communications

# Custom classes includes
from vclass import VClass # For inheritance

class VFarm(VClass):
    def __init__(self) -> None:
        super().__init__()
        # Define the network addresses
        self.servers_addresses = []
        self.own_addresses = {}
        # Prepare the client web-sockets creation (one per server)
        self.servers_websockets = {}

    def __call__(self) -> None:
        try:
```

```
            # Launch the infinite loop
            super().__call__()
            self.verify_arguments()
            asyncio.run(self.connection_handler())
        except KeyboardInterrupt:
            # Cleanly stop the infinite loop

    def verify_arguments(self) -> None:
        # Verify the provided command line arguments and raise errors if needed

    async def connection_handler(self) -> None:
        # Connect to each available server (the CSMs)
        for server_id, server_address in enumerate(self.servers_addresses):
        tasks.append(asyncio.create_task(self.server_handler(server_id, server_address)))
        await asyncio.gather(*tasks)

    async def server_handler(self, server_id, server_address) -> None:
        # Connect to the wanted server (with automatic reconnection)
        async for self.servers_websockets[server_id] in websockets.connect(server_address):
        try:
            task1 = asyncio.create_task(self.incoming_communication_handler(server_id))
            task2 = asyncio.create_task(self.scheduled_communication_handler(server_id))
            tasks = [task1, task2]
            await asyncio.gather(*tasks)
        except websockets.ConnectionClosed:
            # Clean the connection closing and try to reconnect


    async def incoming_communication_handler(self, server_id) -> None:
        # Save the incoming communications and analyze to do the correct followup
        async for message in self.servers_websockets[server_id]:
        result = self.push_on_db(json.loads(message))
        # Analyze the received message and execute the correct followup

    async def outgoing_communication_handler(self, server_id, message) -> None:
        # Send the message to the right server/client
        await self.servers_websockets[server_id].send(json.dumps(message))

    async def scheduled_communication_handler(self, server_id) -> None:
        # Send the initial device information message to the server (CSM)
```

Listing 6.4: VFarm abstract class pseudo code

The VHuman abstract class is a management command line interface for connecting to the Electronic Management System via an instance of a VCsm. This class handles the websocket management and manages the communications between the human and the system. It is mostly used for development and debugging purposes but can also be used to get an inside view of the system. The VHuman class acts as a simple client when looking at the network communications because it only has the possibility to be connected to one server at a time. Moreover, it always sends one message and then wait the device answer. This VHuman class can also be perceived as an example of how to integrate additional external control system to the EMS. In fact, it shows that it is possible to easily interact with the network and thus get and send any wanted information. An external control agent can thus be built on this abstract class (e.g., an external access to a cloud infrastructure). Listing 6.5 shows the pseudo code of the VHuman class.

```
# Libraries includes
import asyncio # For asynchronous communication handling
import websockets # For bidirectional communications

# Custom classes includes
from vclass import VClass # For inheritance

class VHuman(VClass):
    def __init__(self) -> None:
        super().__init__()
        # Define the network addresses
        self.server_address = ''
        self.own_address = ''
        # Prepare the client web-socket creation
        self.own_websocket = None

    def __call__(self) -> None:
        try:
            # Launch the infinite loop
            super().__call__()
            asyncio.run(self.connection_handler())
        except KeyboardInterrupt:
            # Cleanly stop the infinite loop

    async def connection_handler(self) -> None:
        # Connect to the specified server (CSM)
        while True:
            try:
                # Ask what server to connect to
                await self.server_handler()
            except:
                # Catch errors and deal with the known ones

    async def server_handler(self) -> None:
        # Connect to the server (CSM) (with automatic reconnection)
        self.own_websocket = await websockets.connect(self.server_address)
        try:
            while True:
            # Ask what message to send
```

```
            await self.communication_handler(message)
        except websockets.ConnectionClosed:
            # Clean the connection closing and try to reconnect

    async def communication_handler(self, message) -> None:
        # Send the message and wait the response
        await self.own_websocket.send(json.dumps(message))
        response = await self.own_websocket.recv()
```

Listing 6.5: VHuman abstract class pseudo code

The VSensor class is the first member of the last level of inheritance. It is also the last abstract level needed by the electronic management system. That means that the children classes built thanks to this VSensor class will be hardware interactions implementations. As its name implies, the VSensor class is dedicated to sensors interfacing. This abstract class has one abstract method that should be implemented in the hardware interactions source code. This method defines the interface to get a new measurement from the implemented hardware device. Moreover, as it can be seen in Listing 6.6, the VSensor class implements the two abstract methods defined in the VDevice abstract class. As a side note, the compactness and cleanness of this source code shows that the use of the previously detailed abstract classes allow the end source code to be easier to write and better structured.

```
# Libraries includes
import abc # For abstract methods
import asyncio # For asynchronous communication handling

# Custom classes includes
from vdevice import VDevice # For inheritance

class VSensor(VDevice):
    def __init__(self) -> None:
        super().__init__()

    def __call__(self) -> None:
        super().__call__()

    async def scheduled_communication_handler(self) -> None:
        # Send the initial device information message to the csm
        await self.outgoing_communication_handler(self.info)
        # Periodically send data from new measurements
        while True:
            message = await self.new_measurement()
            await self.outgoing_communication_handler(message)
            await asyncio.sleep(self.sleep_time)

    async def new_measurement(self) -> json:
        # Provide the abstract interface to ask a new measurement
        new_measurement = self.get_new_measurement()
        return new_measurement

    @abc.abstractmethod
    async def get_new_measurement(self) -> json:
        # Abstract interface to get a new measurement
```

Listing 6.6: VSensor abstract class pseudo code

The VActuator class is the second member of the last level of inheritance. It is very similar to the VSensor class but is dedicated to actuators. As it can be seen in Listing 6.7, the difference between sensors and actuators hardware interaction interface is marked here by the abstract method definition. In fact, the VActuator class defines the “set_new_status” methods whereas the VSensor class defines the “get_new_measurement” abstract method. Otherwise, all the remarks made in the VSensor class description are also valid for this VActuator class.

```
# Libraries includes
import abc # For abstract methods
import asyncio # For asynchronous communication handling

# Custom classes includes
from vdevice import VDevice # For inheritance

class VActuator(VDevice):
    def __init__(self) -> None:
        super().__init__()

    def __call__(self) -> None:
        super().__call__()

    async def scheduled_communication_handler(self):
        # Send the initial device information message to the csm
        await self.outgoing_communication_handler(self.info)
        # Periodically send data from new measurements
        while True:
            message = await self.new_measurement()
            await self.outgoing_communication_handler(message)
            await asyncio.sleep(self.sleep_time)

    async def new_measurement(self):
        # Provide the abstract interface to ask a new measurement
```

```
        new_status = self.set_new_status()
        return new_status

    @abc.abstractmethod
    async def set_new_status(self):
        # Abstract interface to set a new status
```

Listing 6.7: VActuator abstract class pseudo code

As it can be understood from the pseudo codes presented throughout this section, some decision making algorithm (being basic automation or advance cognitive computing) can be implemented in multiple places: in the incoming communication handlers (before and after having analyzed the messages), in the outgoing communication handlers (before and after having sent the messages), in the scheduled communication handlers (when sending a new message or between messages), and in the "new_measurement" method that is called every time a new measurement is wanted. As it is seen in the next section, only basic automation algorithms are currently implemented via the scheduled communication handlers. These basic algorithms are based on timers and other sensors measurements values to activate or deactivate a device (e.g., the water pump activate itself for 5 minutes every 15 minutes).

6.1.2 Hardware interactions

Concerning the hardware interactions classes, they all rely on the control system classes. More precisely, as it can be noticed in Figure 6.3, they rely on the VSensor and Vactuator abstract classes. In fact, each hardware interaction object is a realization of a class that implements the needed abstract methods and fields.

Seeing that the hardware interactions classes are pretty similar to each other, only three of them are presented hereafter: a sensor, an actuator, and a probe. Each one presents some particularities that can be found in other devices of the same type and some similarities that are common to every VDevice object.

The first presented hardware interaction class is the AM2315C sensor. As it was presented in Section 5.1, it is an I²C temperature and humidity sensor. The pseudo code of its hardware interaction class is presented in Listing 6.8 and four main characteristics can be viewed. Firstly, a path variable is defined on the first lines. This is made to allow this piece of code to find the source code for the EMS control system abstract classes. Secondly, the initialization of the sensor calls the sensor library and the sensor class itself inherits from the VSensor abstract class. Thirdly, the abstract method "get_new_measurement" is implemented thanks to the sensor library. And fourthly, a main function is present. This entry point instantiates the sensor and launches its infinite loop.

```
# Path variables
import sys # For system interactions
sys.path.insert(0,'../BASE_CLASSES') # For base class loading

# Libraries includes
import am2315c_library # For the AM2315C sensor

# Custom classes includes
from vsensor import VSensor # For inheritance

class AM2315C(VSensor):
    def __init__(self) -> None:
        # Initialize the sensor id and bus
        super().__init__()
        self.id = '1'
        self.unique_id = 'AM2315C' + '_' + self.id
        self.device_info = {'type': self.SENSOR, 'id': self.unique_id}
        self.sensor = am2315c_library.AM2315(powerpin=26, bus=1)
        self.sleep_time = 30

    def __call__(self) -> None:
        super().__call__()

    def get_new_measurement(self) -> json:
        # Implement the get_new_measurement abstract method
        results = self.sensor.read_humidity_temperature_crc()
        new_measurement = {'humidity': results[0], 'temperature': results[1]}
        new_measurement['time'] = self.get_formatted_time()
        new_measurement['date'] = self.get_formatted_date()
        return new_measurement

```

```
if __name__ == '__main__':
    # Instantiate and execute the sensor infinite loop
    am2315c = AM2315C()
    am2315c()
```

Listing 6.8: AM2315C hardware interaction class pseudo code

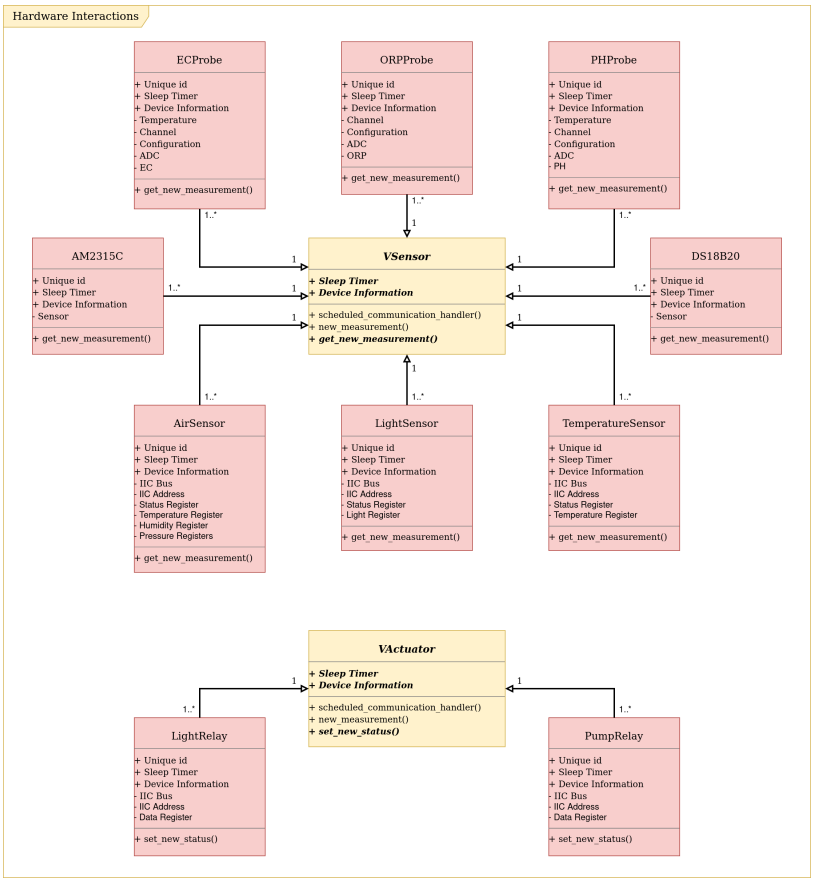

Figure 6.3: Class diagram of the hardware interactions source code that compose the EMS. The electronic control system high level abstract classes are color-coded in yellow while the hardware interactions classes are represented in red (because they directly interact with the hardware).

The second presented hardware interaction class is the water pump relay. This pump is activated at regular intervals and is able to correctly operate without any network connection thanks to its time based local computations. This edge computing system is an example of how to include automation algorithms in the device itself. Moreover, this class implements the "set_new_status" method that is defined by the VActuator abstract class. This is the main difference with the previously shown sensor but, alongside these particularities, the three other remarks written above also apply: the path variable is also defined, the initialization uses a library (this time, the I^2C one), and the main entry point is also present. All of these can be found in Listing 6.9.

```
# Path variables
import sys # For system interactions
sys.path.insert(0,'../BASE_CLASSES') # For base class loading

# Libraries includes
import smbus # For I2C connection

# Custom classes includes
from vactuator import VActuator # For inheritance

class PumpRelay(VActuator):
    def __init__(self) -> None:
        # Initialize the actuator id and bus
        super().__init__()
```

```
        self.id = '1'
        self.unique_id = 'PumpRelay' + '_' + self.id
        self.device_info = {'type': self.ACTUATOR, 'id': self.unique_id}
        self.sleep_time = 0
        self.i2c_address = 0x10
        self.i2c_register = 0x01
        self.i2c_bus = smbus.SMBus(1)

    def __call__(self) -> None:
        super().__call__()

    def set_new_status(self) -> json:
        # Implement the set_new_status abstract method
        current_time = datetime.datetime.fromtimestamp(current_time, tz=None)
        if current_time.minute % 15 < 5:
            delay = ((4 - (current_time.minute % 5)) * 60) + (60 - current_time.second)
            new_measurement = {'device': 'pump', 'status': 'on', 'duration': delay}
            self.i2c_bus.write_byte_data(self.i2c_address, self.i2c_register, 0xFF)
            self.sleep_time = delay
        else:
            delay = ((14 - (current_time.minute % 15)) * 60) + (60 - current_time.second)
            new_measurement = {'device': 'pump', 'status': 'off', 'duration': delay}
            self.i2c_bus.write_byte_data(self.i2c_address, self.i2c_register, 0x00)
            self.sleep_time = delay
        new_measurement['time'] = self.get_formatted_time()
        new_measurement['date'] = self.get_formatted_date()
        return new_measurement

if __name__ == '__main__':
    # Instantiate and execute the actuator infinite loop
    pump_relay = PumpRelay()
    pump_relay()
```

Listing 6.9: PumpRelay hardware interaction class pseudo code

The last hardware interaction class that is presented is the pH probe. Alongside the path variables definition, the "get_new_measurement" implementation, and the main entry point, three other particularities can be observed in the pseudo code presented in Listing 6.10. Firstly, the initialization function is in charge of loading and verifying the configuration file that contains the probe calibration values. Secondly, the defined class uses two libraries: the ADC and the pH libraries. Thirdly, a calibration method is defined. This additional method is a wrapper for the probe calibration system defined in the pH library.

```
# Path variables
import sys # For system interactions
sys.path.insert(0,'../BASE_CLASSES') # For base class loading

# Custom classes includes
from vsensor import VSensor # For inheritance
from ph_library import PH # For the PH library
from adc_library import ADC # For the ADC library

class PHProbe(VSensor):
    def __init__(self) -> None:
        # Initialize the probe id and bus
        super().__init__()
        self.id = '1'
        self.unique_id = 'PHProbe' + '_' + self.id
        self.device_info = {'type': self.Sensor, 'id': self.unique_id}
        self.sleep_time = 30
        self.temperature = 20.0
        self.channel = 2
        # Get the probe calibration information
        self.config_path = os.getcwd() + '/ph_config.ini'
        self.config = self.get_config(self.config_path)
        if not self.config:
        self.raise_file_error(self.config_path, 'PHProbe')
        else:
        # Verify the probe callibration information
        keys = ['neutral_volt', 'acid_volt', 'ref_offset']
        for key in keys:
        if not key in self.config:
        self.raise_config_error(key, 'PHProbe')
        # Instantiate the needed objects
        self.adc = ADC()
        self.ph = PH(self.config['neutral_volt'], self.config['acid_volt'], self.config['ref_offset'])

    def __call__(self) -> None:
        super().__call__()

    def get_new_measurement(self) -> json:
        # Implement the get_new_measurement abstract method
        voltage = self.adc.read_voltage(self.channel)
        result = self.ph.readPH(voltage)
        new_measurement = {'ph': result, 'unit': 'pH'}
        new_measurement['time'] = self.get_formatted_time()
        new_measurement['date'] = self.get_formatted_date()
        return new_measurement

    def do_calibration(self) -> dict:
        # Define the calibration wrapper
        voltage = self.adc.read_voltage(self.channel)
        calibration = self.ph.calibrate(voltage)
        if not 'error' in calibration:
            self.update_config(self.config_path, calibration)
        return calibration

if __name__ == '__main__':
```

```
# Instantiate and execute the probe infinite loop
ph_probe = PHProbe()
ph_probe()
```

Listing 6.10: PHProberobe hardware interaction class pseudo code

In addition to the already discussed basic automation system present in these hardware interaction classes, another particularity all these implementations have in common is noteworthy. By looking at the presented pseudo codes (Listings 6.8, 6.9, and 6.10), it can be seen that the class variable "sleep_time" is present in each initialization function. As its name suggests, this variable defines the time a device will sleep between two scheduled measurements. It may seem useless but, in reality, the presence of this variable shows that the device itself can decide when to sleep and when it should do something. Thanks to this particularity, the device automation can be fully managed by the device itself and this timer can be used to trigger automation algorithms.

Most sensors and actuators come with a datasheet that can be used to have an idea of how to operate the device. Thanks to the EMS control system classes and with the datasheet information, the development of the hardware interaction codes can be straightforward. However, keeping an organized code base can be a real challenge with the multiplication of electronic devices and with various requirements each sensor and actuator needs (e.g., proprietary libraries, custom communication protocols, etc.). Thankfully, a folder based approach can solve this problem as long as the relative path to the control system classes is known. The used code organization is described in Appendix A and the electronic devices composing the current hardware assembly POC are already described in Section 5.1.

However, some hardware devices are harder to interact with. This can originate from multiple reasons: the used hardware (e.g., some HAT make devices harder to use), the communication protocols (e.g., the I^2C protocol is less practical to use than a direct GPIO connection), the device itself (e.g., the functioning of a temperature sensor should be easier that the one of a pH probe), etc. Thankfully, these problems are (and should be) often managed in the device library. The hardware interaction classes can thus be fairly clean, structured, and standardized. As a side note, multiple remarks can be made to explain the problems encountered during the implementation of the current hardware assembly POC (refers to Appendix A for more details):

- The pH, EC, and ORP probes each need their own library because of the computations required to transform the raw voltage into the wanted measurement. These computations need to take the probe calibration values into account. Additionally, the use of an ADC HAT that requires its own library was needed to interface these probes with the RPI.
- The AM2315C humidity and temperature sensors require their own library to compute a checksum to verify the measurement correctness. Moreover, a custom I^2C implementation has to be used to correctly interface these sensors with the RPI.

6.2 Management dashboard

The Management Dashboard is the cross-platform, dynamic, and responsive application developed for the SPF human operators to view, manage, and interact with the SPFs. Before explaining its functioning, the major technical details can be listed. As for the EMS, it is worth knowing that this MD is the result of the technical choices described in Sections 4.3 and 4.4. If needed, additional details (i.e., line counts, external authors copyright, etc.) about these source code and a repository link can be found in Appendix A:

- Dart Software Developer Kit (SDK) 2.17 or above with the Flutter Framework 3.0 or above.
- The Flutter MaterialUI package "material.dart", the Flutter table icon package "flutter_tabler_icons.dart", and the Syncfusion Flutter chart package "syncfusion_flutter_charts.dart".

The MD is cross-platform thanks to the use of the Flutter framework. In fact, if the used packages are wisely chosen (which is the case for this master thesis), the application can be executed on these platforms: IOS, Android, Web Browsers, Linux, Windows, and MacOs. Additionally, the MD is responsive. That means that is can adapt the displayed elements to the size of the used screen (and it can even do it dynamically when resizing windows). Moreover, the application also is dynamic. That means that the displayed buttons, tiles, and the content information are different according to the used platform (i.e., smartphone, tablet, or desktop screen). Finally, the MD built is entirely modular. In fact, this is a inherent feature of the Flutter framework. For this project, this is an important feature because both developments (i.e., the EMS and the MD) are fully object oriented thus making them very modular and easy to modify (e.g., to add a new page, to modify the configuration file parser, etc.).

While the next section is dedicated to show these particularities (Section 6.2.1), the following one is more focused on the internal structure of the MD source code (Section 6.2.2). However, before going deeper into the analysis, one precision needs to be made: with the Flutter framework, the objects are called "Widgets". In fact, a widget can be stateful (e.g., a text input that dynamically analyzes the entered text) or stateless (e.g., a page that does not adapt according to the user input). In this MD implementation, most of the widgets are stateless because the dynamism and responsiveness do not need stateful widgets to be implemented. More precisely, the only widgets that need to be stateful are the buttons, text fields, and graphs.

6.2.1 Design and functionalities

On the design side, the Management Dashboard (MD) is fairly basic. Still, the displayed elements are pretty, clean, and simple. A global theme is followed in all the pages and all buttons are easily identifiable. The main design effort was put on the devices support: smartphone, tablet, and desktop. Seeing that the application is responsive and dynamic, some design elements vary according to the used device. For example, a side menu is always visible in tablet and desktop mode while it has to be slided in front of the screen in smartphone mode. The opposite system is in place for a top menu, it is only visible in smartphone mode. Additionally, the same principle is implemented for some buttons with the objective of not having redundant actions on the screen. These particularities can be seen by comparing Figures 6.4, 6.5, and 6.6. Firstly, the responsiveness can be perceived by looking at the green, orange, and red tiles (their width adapt according to the available space). Secondly, the dynamism is demonstrated by the presence or absence of the blue buttons in these same tiles (these buttons are not needed in desktop mode). Thirdly, the two menus are visible only on the needed devices. And fourthly, the design is coherent and simple across all devices.

On the functionality side, the Management Dashboard (MD) is full-fledged. All the wanted functionalities are implemented. The Management Dashboard (MD) includes:

- A summary page showing the available SPFs (Figure 6.4).
- A page detailing the information and metrics of the selected SPF (Figure 6.9).
- A summary page regrouping all the CSMs of the selected SPF (Figure 6.10).
- A page detailing the information and metrics of the selected CSM (Figure 6.11).
- A summary page listing all the devices of a specific CSM of the selected SPF (Figure 6.12).
- A page detailing the information, configuration, and measurement history of a given device of a specific CSM from the selected SPF (Figure 6.13).
- Interactive graphs for metrics and measurements history (Figures 6.9, 6.11, 6.13, and 6.14).
- A text form for modifying the information and configuration of the selected device (Figure 6.14).

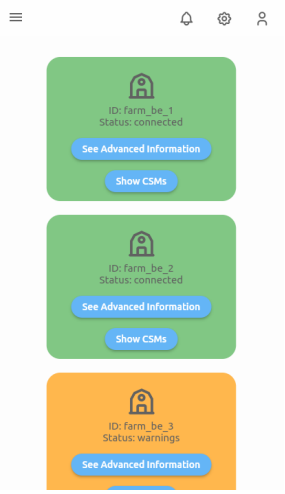

Figure 6.4: The smartphone version of the farms summary page (FarmBody) of the farm view of the MD.

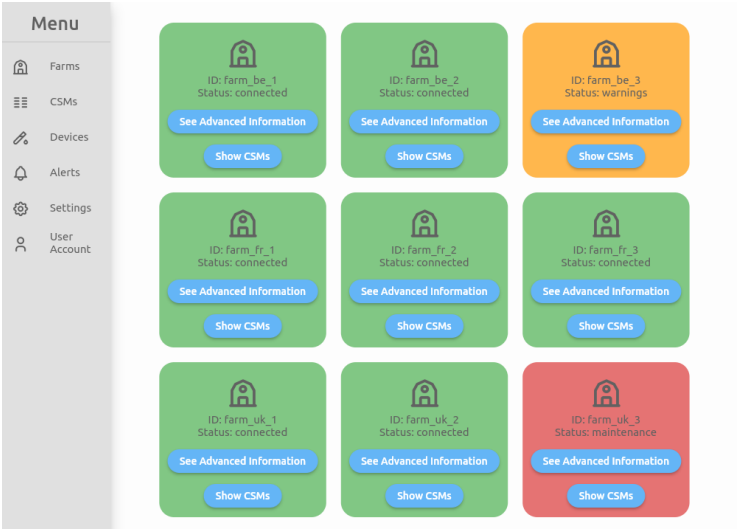

Figure 6.5: The tablet version of the farms summary page (FarmBody) of the farm view of the MD.

Figure 6.6: The desktop version of the farms summary page (FarmBody) of the farm view of the MD.

- A menu for selecting the wanted view in addition to the user account menu, general settings, and the received notifications (Figure 6.8 for the smartphone mode and Figures 6.5 and 6.6 for the tablet and desktop mode).

In addition to all these functionalities, the application navigation is very easy and offers many shortcuts. All the relations between pages can be seen in Figure 6.7. Each link on this relational diagram means that there is an easy way to go from one page to the other (e.g., with a button or by clicking on a tile). This diagram also shows that the application is composed of three main views: the farm or SPF view, the CSM view, and the device view. Each view is focused on one type of component (respectively the SPFs, the CSMs, and the sensors and actuators). Moreover, each view offers two pages: a summary and a detailed page. Finally, the device view includes an additional page to modify and update the devices configuration: the update page.

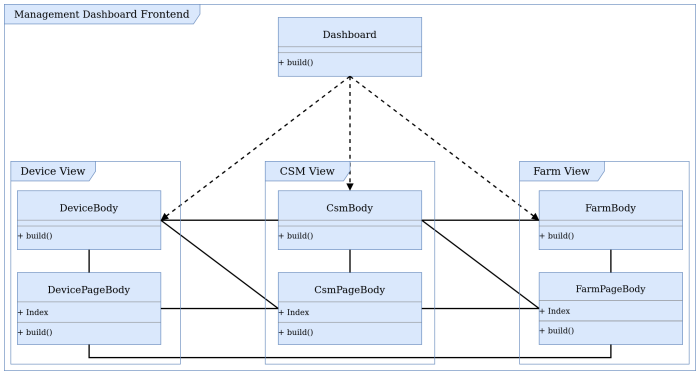

Figure 6.7: Relational diagram of the MD frontend views.

Concerning the first view, the farm or SPF one, its pages are presented in Figures 6.4 and 6.9. As it can be seen, the farms summary page is composed of a grid of tile. Each tile is an available SPF and it is colored according to the SPF status. Under each SPF icon, the id and the status are shown. Additional, two action buttons are present in each SPF tile: the first one moves the user from the summary page (Figure 6.4) to the detailed page (Figure 6.9) while the second one moves the user to the CSMs summary page (Figure 6.10). The farm details page (Figure 6.9) is composed of a status bar, a farm information box, and multiple graphs. These graphs are interactive, are generated according to the available data, and allow the user to directly access the concerned device detailed page. Moreover, the status bar is also colored according to the SPF status and it contains an action button to go back to the farms summary page (Figure 6.4).

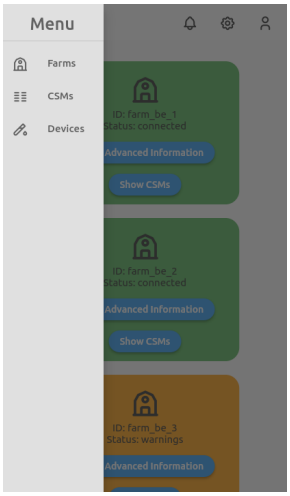

Figure 6.8: The smartphone version of the side menu of the MD.

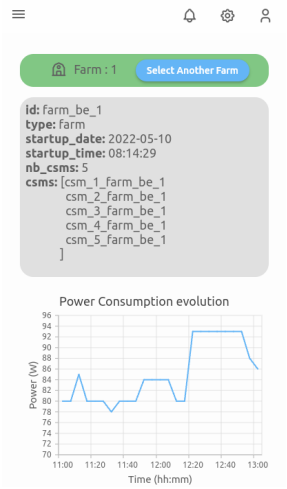

Figure 6.9: The smartphone version of the farms detailed page (FarmPageBody) of the farm view of the MD.

Figure 6.10: The smartphone version of the CSMs summary page (CsmBody) of the CSM view of the MD.

Figures 6.10 and 6.11 present the two pages of the CSM view. The first one is the CSMs summary page that is composed of a grid of tile where each tile is a CSM with its id and status. Similarly to the farms summary page, each tile also contains an icon and the two action buttons. Additionally, this CSMs summary page also contains a status bar that informs about the selected SPF, its status, and allows the user to go back to the farm summary page (Figure 6.4). The second

page of the CSM view is the detailed CSM page. It provides information about the current CSM and presents the monitored metrics in graphs. As for the farms detailed page, this CSM detailed page also contains a status bar and some actions buttons. Moreover, the graphs are also interactive and allow the user to directly access the details of the concerned device.

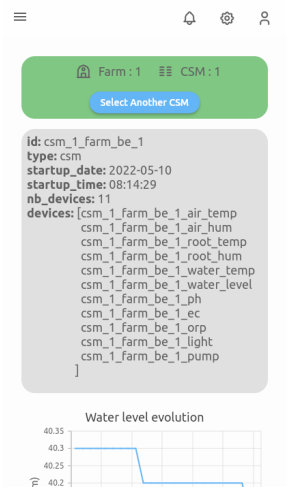

Figure 6.11: The smartphone version of the CSMs detailed page (CsmPageBody) of the CSM view of the MD.

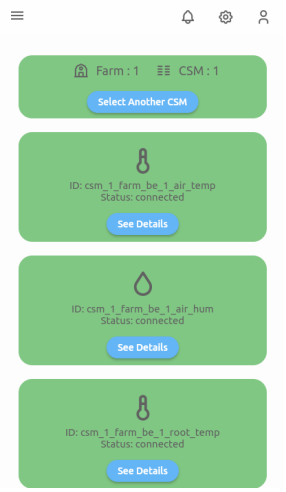

Figure 6.12: The smartphone version of the devices summary page (DeviceBody) of the device view of the MD.

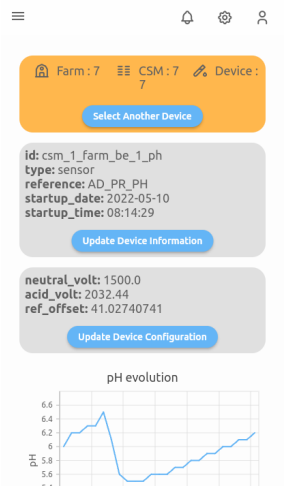

Figure 6.13: The smartphone version of the devices detailed page (DevicePageBody) of the device view of the MD.

The third view is the device view. It can also be called the sensor and actuator view. It is presented in Figures 6.12, 6.13, and 6.14. The two first pages (Figures 6.12 and 6.13) of this view follow the same logic than the two pages of the two other views. One particularity of the devices summary page is that the icons are dynamic to better reflect the type of sensor or actuator. Additionally, the detailed device page shows the device information but also the device configuration. Moreover, these two text boxes are editable thanks to the update page that is presented in Figure 6.14. This last page is a multi-line text form that is able to directly modify the device information and configuration.

Figure 6.14: The desktop version of the update configuration page (UpdateBody) of the device view of the MD.

To end this MD functionalities overview, two points can be highlighted: the grid and graph dynamism, and the desktop mode displaying. For the first point, the application responsiveness was pushed to its maximum potential by adapting the grid size dynamically (this behavior can be observed by comparing Figures 6.4 and 6.5). In the same way, the graphs available in all detailed pages also adapt to the width of the screen by displaying more historical data (this behavior can be observed by comparing Figures 6.13 and 6.14). Concerning the desktop mode displaying, it can be seen in Figures 6.6 and 6.14 that the desktop devices get a double pages displaying. In fact, this was made to optimize the available space thanks to the MD responsiveness. More precisely, the two adjacent pages are always in relation with each other: Figure 6.6 shows the farms summary page on the left with the farm detailed page on the right and Figure 6.14 shows the device detailed page on the left with the device update page on the right. Obviously, this system is also available for the other pages of the application.

6.2.2 Internal structure

Concerning the internal structure of theMD, the three main views presented in the previous section (Section 6.2.1) can be found. The class diagram of each view is presented in this section. Each one is also reviewed and some interesting pseudo code are presented.

Before presenting the three main views, the common widget tree can be explained. The base widgets that compose this common widget tree are dependent of the used framework: the DashboardApp widget is a realization of the Dashboard widget that is itself a realization of other multiple widgets. This Dashboard widget is composed of a SideMenu (i.e., the drawer), a TopMenu (i.e., the appbar), and a body widget. This body widget is different for each of the three main views that are reviewed hereafter. This can be seen in Figure 6.15.

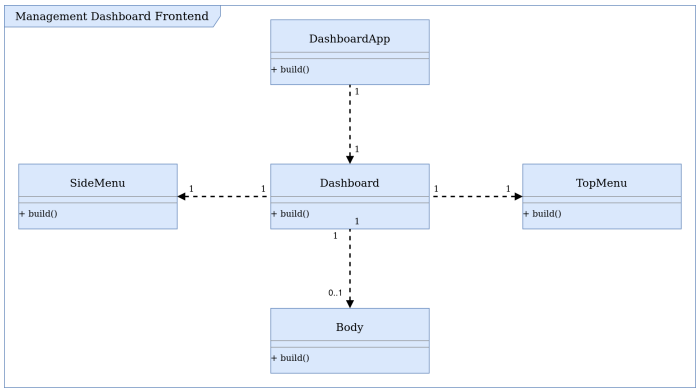

Figure 6.15: Widget diagram of the MD frontend entry point.

The first page that is presented is the farm view. The widget diagram of this view is presented in Figure 6.16. This farm view regroups two pages: the FarmBody and the FarmPageBody. These pages are composed of multiple sub-widgets: the FarmBody is composed of a FarmGrid that is itself a composition of FarmTile and the FarmPageBody is composed of a FarmPage that is itself a composition of FarmInfoTile and GraphTile. In desktop mode, these two pages are identical. However, in tablet and smartphone mode, each page prioritizes one sub-widget: the FarmBody prioritizes the FarmGrid and the FarmPageBody prioritizes the FarmPage. This prioritization system is implemented to have a responsive application that is dynamic. In fact, displaying the same information in smartphone, tablet, and desktop mode leads to redundancies (e.g., a button to select another farm is not needed if the list of available farms is displayed alongside the current farm information). Having this dual page system thus provides a better user experience.

In the farm view, two widgets are interesting: the FarmGrid and the GraphTile. The fisrt one is presented in Listing 6.12 and is interesting because the grid of available farms is dynamically generated according to the number of known farm. The second one is visible in Listing 6.11 and is interesting because it is a stateful widget that implements an interactive graphical representation of a metric (e.g., the air temperature). The graph data is dynamically generated and multiple properties of the graph can be modified to make it more interactive.

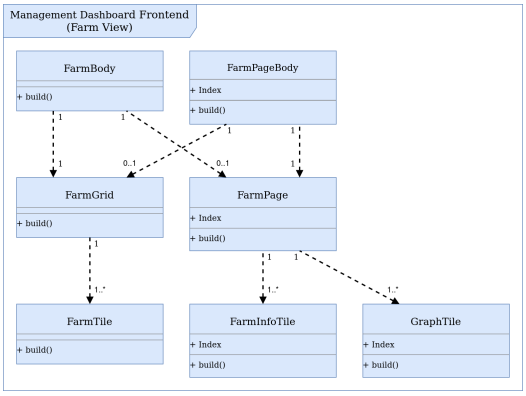

Figure 6.16: Widget diagram of the farms pages of the MD frontend.

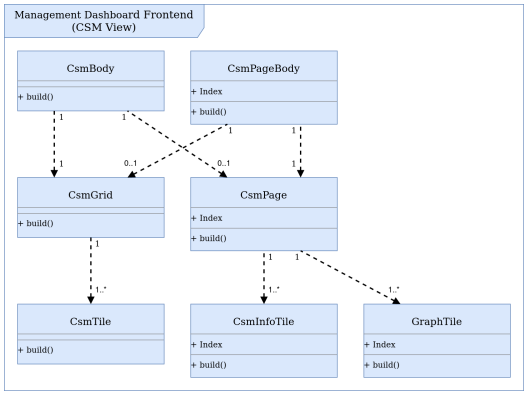

Figure 6.17: Widget diagram of the CSMs pages of the MD frontend.

```
class GraphTile extends StatefulWidget {
    @override
    _GraphTileState createState() {
        return _GraphTileState(index: index);
    }
}

class _GraphTileState extends State {
    _GraphTileState({required this.index});

    @override
    void initState() {
        _tooltipBehavior = TooltipBehavior(enable: true);
        super.initState();
    }

    @override
    Widget build(BuildContext context) {
    return SfCartesianChart(
        tooltipBehavior: _tooltipBehavior,
        title: // Set graph title,
        primaryXAxis: // Set graph axis properties,
        primaryYAxis: // Set graph axis properties,
        series: [
            LineSeries(
            dataSource: List.generate(
                GraphData.time[index].length,
                (offset) => ChartData(GraphData.time[index][offset],
                double.parse(GraphData.value[index][offset]))),
            xValueMapper: (ChartData data, _) => data.x,
            yValueMapper: (ChartData data, _) => data.y)
            ],
        );
    }
}
```

Listing 6.11: GraphTile widget pseudo code

```
class FarmGrid extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
    return Wrap(
        children: [
            for (var index = 0; index < FarmData.nb_farm; index++)
                FarmTile(index: index),
            Container(/*Define the top status bar*/);
    }
}
```

Listing 6.12: FarmGrid widget pseudo code

The class widget diagram presented in Figure 6.17 corresponds to the second page: the CSM view. This view is very similar to the far view. Its widget tree is identical excepted that all widgets are adapted for CSM display. Moreover, some widget content is different (e.g., the CsmInfoTile

is formated differently than the FarmInfoTile). The prioritization system and the application responsiveness and dynamism are also identical.

The last presented page is the device view. This view is marginally different from the two previously mentioned views. It is presented in Figure 6.18 and the differences can be spotted at the bottom of the widget diagram. The fact that the DevicePage widget is composed of three widgets is the result of the calibration requirement of some devices. This is the reason why the DeviceInfoTile and the DeviceConfigTile are themselves a realization of an UpdateTile widget. This last widget is interesting because it is a stateful widget that implements a multi-line text from field. The user can use this input field to write and upload a device configuration that directly modify the behavior of the device. The pseudo code of this widget can be seen in Listing 6.13.

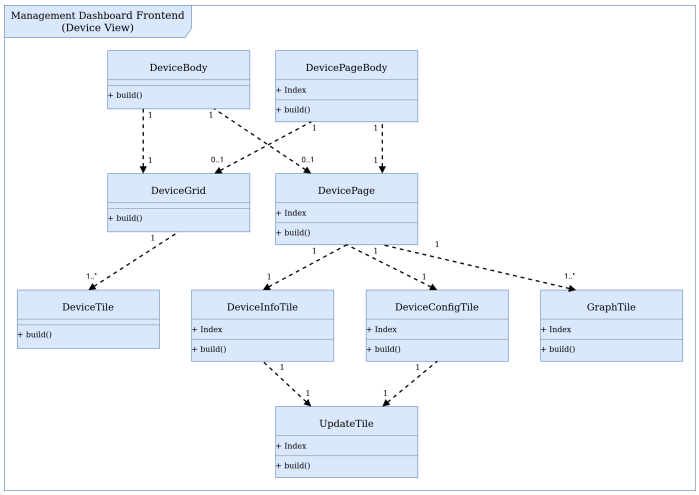

Figure 6.18: Widget diagram of the devices pages of the MD frontend.

```
class UpdateTile extends StatefulWidget {
    @override
    _UpdateTileState createState() => _UpdateTileState();
}

class _UpdateTileState extends State {
    @override
    Widget build(BuildContext context) {
    return Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
            Container(
                // Define the container characteristics
                child: Column(
                    children: [
                        TextFormField(/*Define the text form*/),
                        Row(
                        children: [
                            ElevatedButton(/*Define the submit button*/),
                            ElevatedButton(/*Define the cancel button*/),
                        ],
                        ),
                    ],
                ),
            ),
        ],
        );
    }
}
```

Listing 6.13: UpdateTile widget pseudo code

To end this section reviewing the source code, it is interesting to understand why each and every piece of software were not described. The important source code were detailed with a pseudo code and various diagrams but this could not be applied to all implemented classes. Doing so would have overcharged this report while not improving the quantity of useful information. However, to get further information about the source code, special attention should be paid to Appendix A. Moreover, a link towards the repository is also provided to allow a deeper dive into the code base if wanted.

Chapter 7

Results

To test, validate, and use the hardware and software developments discussed in Chapters 5 and 6, a real work cultivation system was set-up. This cultivation tray accommodates the hardware assembly that is itself used to host the developed software pieces. This section reviews the obtained results from testing the hardware and software builds on this cultivation system.

The real world cultivation system developed to deploy the hardware and software sides of the AIP is presented in Figure 7.1. It is a small scale cultivation tray that is built out of wood studs and with plastic containers. It may seem a bit rough at first glance but this cultivation chamber was built with many constraints and still maintains the useful functionalities.

Figure 7.1: The real world cultivation system developed to deploy the hardware and software sides of the AIP.

Regarding the constraints encountered during the crafting of this cultivation tray, the major ones are: space, time, cost, and knowledge. The space and cost constraints are linked together and derive from the nature of this project. Because it is an end of study project and not an enterprise project, the space and money resources were limited. Concerning the time limitations, it comes from the fact that this build is only a test bed for this master thesis project. It is thus not well valued but has to be built to wrap-up this project. Finally, the absence of prior knowledge has made the construction of this cultivation system much harder than it might seem. A deep analysis of the systems discovered in the literature was necessary. As a side note, the presented cultivation chamber is the third evolution of the first design. Since the beginning of this project, some leaking, lighting, and nutrient dosage problems were encountered and solved thanks to the read literature.

The main features of this small scale cultivation tray are: independent nutrient solution reservoir and cultivation tray, a modular number of cultivation chambers, an aero-hydroponic system, a vertical adjustment for the light panel, and a top floor available for the electronic and computing equipment. This last feature is linked to the space constraint but is also useful to secure the electronic computing elements from the water sources. Concerning all the other features, their only purpose is the ease of use of the system and its efficiency to grow plants. If wanted, Appendix B shows some data from cultivation experiments made during this project realization.

With the test setup that is this small scale cultivation system, the POC of the electronic assembly can be deployed in the reservoir and cultivation chamber. Obviously, the pump is located in the reservoir and the light panel (Figure C.10) is positioned over the growing plants. The water quality probes (Figures C.7, C.8, and C.9) are used in the nutrient solution reservoir as well as one waterproof temperature sensor (Figure C.2). Finally, the other waterproof temperature sensor is in the root zone of the cultivation chamber and the two temperature with humidity sensors (Figure C.1) are placed between the plants and the lighting system (to monitor the difference of temperature and humidity between the growing tray and the ambient air of the room). Finally, because of the physical distance between the reservoir and the cultivation tray, two RPIs were deployed to avoid having a wiring mess. This deployed system is visible in Figures 5.1 (wiring diagram) and 7.2 (deployment diagram).

Once the hardware developments were installed, the MVP of the software side of the AIP could be deployed. To better mimic a real word experiment, a third RPI was deployed to host the VCsm and VFarm children classes as well as the MD frontend and backend. The two previously mentioned RPIs are thus used to host the needed VSensor and VActuator children classes. This deployment is visible on the diagram in Figure 7.2.

With this fully featured test-bed, the best tests to perform to assess the quality of the functionality of the developed AIP is to cultivate some plants. Such a cultivation experiment was done and its resulting data are visible in Appendix B. In fact, multiple generation of lettuce were cultivated. Unfortunately, some tests went well and some other did not. However, the reasons some tests failed are not linked to the developed AIP but to the limitations of the cultivation chamber. To be more precise, the hardware and software side of the AIP work flawlessly (after some initial debugging and some generational improvements through time and experiments). The main causes of the test failure are: water splitting, temperature control, and cultivation knowledge. For the first one, the lettuce leaves were partially splashed with water coming from the aero-hydroponic system. With the addition of a powerful light panel, this causes some burns to the plant leaves. For the second one, the developed cultivation chamber does not have any temperature control. Additionally to the fact that the cultivation system is located in a basement, some months are not ideal to try cultivate lettuce (e.g., during the winter). Moreover, the lack of temperature control is also visible in the root zone of the growing chamber. This time, the effect is in the opposite way: the light panel rises the root zone temperature above the ideal growing conditions. Finally, concerning the third reason, the lack of prior knowledge in the field makes it very hard to diagnose plants diseases. For example, one harvest was completely destroyed by aphids.

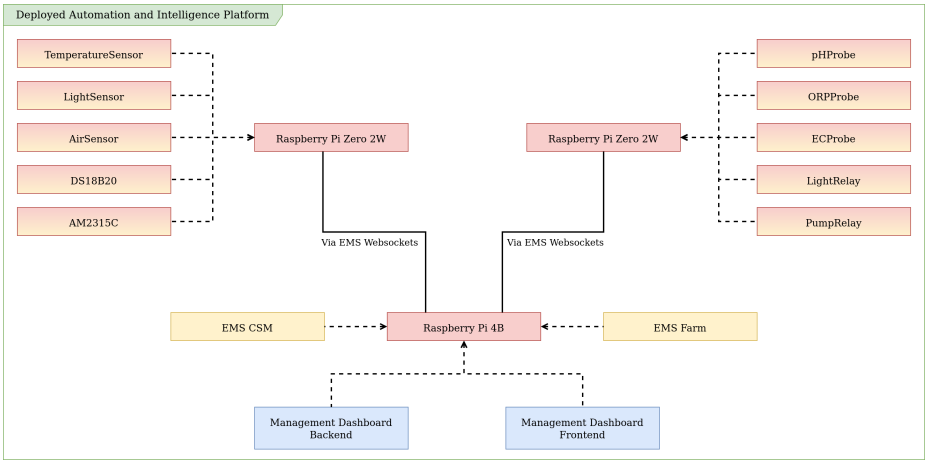

Figure 7.2: Deployment diagram of the SPF AIP. The electronic system hardware is color coded in red, the EMS control system in yellow, the EMS hardware interactions with a gradient from red to yellow (because the code source and the electronic components share the same names), and the MD in blue.

The outcomes of the performed tests are thus pretty positive and encouraging. In fact, the final version of the developed AIP worked as expected and the solutions to the encountered environmental issues can all be addressed by additional environmental controls fully compatible with the current AIP. These possible improvements and some additional hardware and software tests are discussed in the next chapter, Chapter 8. Moreover, some improvements of the cultivation techniques that shows the usefulness of the AIP can be observed in the appendix B.

Chapter 8

Discussion

To finalize the development of the Automation and Intelligence Platform (AIP), a retrospective and perspective analysis of the study, development, and tests performed during the realization of this master thesis is made. For each step of the development, the Minimum Viable Product (MVP) of the control software and the Proof Of Concept (POC) of the monitoring hardware are reviewed. This two-sections chapter begins with the project retrospectives and ends with the future works perspectives.

8.1 Retrospective analysis

Beginning with the POC of the monitoring hardware, the main retrospective remark to make is about the testing phase. In fact, the research step was pretty fruitful thanks to the finding of some similar projects in the scientific literature and the hobbyist blog posts. Similarly, the development phase was proactive seeing that the built POC fulfills the basic capabilities needed to assess the good functioning of the AIP. Moreover, this stage of the project was complemented with the design of a more advanced electronic system that implements all the basic requirements of a Smart Plant Factory (SPF). However, the testing step of the POC of the monitoring hardware showed plants growing difficulties due to the lack of control over the external environment and to the lack of experience for building plant growing trays. As it can thus be deduced, in addition to the development unit testing, the real world growing experiments were the most important tests conducted for assessing the proper working of the monitoring hardware POC (additional pieces of information are available in Appendix B). Compared to other electronic systems found in the literature, this master thesis monitoring hardware build is more modular, better documented, compatible with the edge and cloud computing paradigms, and still implements all the same functionalities. No major drawbacks were detected during its testing and the only slightly annoying aspects of the electronic assembly are the amount of components needed to make the system modular (e.g., I^2C multiplexers) and the non native 5V compatibility of the chosen RPI platform.

Then, concerning the MVP of the control software, the retrospective analysis is more complex. Firstly, the read scientific papers were very disparate in terms of subject and in terms of technical descriptions. In fact, some reports were also focused on the development of similar projects while not including any real technical documentation and other studied the ICT architectures suitable to SPFs deployment while not implementing a real world system. This master thesis did not take advantage of neither of those because of its multisectoral focus and its technical hardware oriented technical developments. Secondly, the studied implementations and the made developments are both significant (more details are available in Appendix A) but not fully completed. The major shortcomings of the developed control software are the lack of a proper multi-node communication routing algorithm in the Electronic Management System (EMS) and the absence of full fledged backend implementation for the Management Dashboard (MD). Thirdly, the testing step of the MVP of the control software was the most flawless part of the project thanks to the efficiency of the

real world growing experiments. Compared to the software control systems found in the literature, the developed one is much more technically documented, has some headroom for improvements, can be deployed on an arbitrary number of SPFs, and is completely modular when considering the hardware interactions. On the negative side, the implementation time and the non production ready backend server are its main shortcomings.

Finally, about the developed test cultivation tray, some afterthoughts can be mentioned. On the literature review side, most of the found growing systems are more advanced and larger than the developed one. However, this custom cultivation chamber is a good fit for this master thesis and it has some room for interesting improvements development (e.g., to test other types of Hydroponics Systems). For the development step, the two previous iterations of this work were worse than the current status. The currently used system is functional and was sufficient to perform important testings. Talking about this last phase of the project, the tests realized thanks to this cultivation tray were the most important. They enabled the in-depth testing of the MVP and POC. In addition to the highlighting of shortcomings that were solved iteratively, their outcomes show the pros and cons of each component.

8.2 Future works

Regarding the perspectives of the developed Automation and Intelligence Platform (AIP), the range of possible improvements and additional capabilities is wide. Starting at the lowest technical level, the followups to the cultivation tray are:

- Increase the environmental isolation of the culture chamber from the external environment to provide a better and more stable growing environment to the plants.
- Improve the cultivation tray by solving the shortcomings highlighted by the performed testing (e.g., water splashes, nutrient solution reservoir size, etc.).
- Remove the testing setup and increase the plant density by using the available space in a more optimized format.

Then, for the monitoring hardware POC, the possible improvements are:

- Build the presented electronic system advanced design to assess their proper working and to improve the monitoring and automation capabilities.
- Standardize the assembly to the cultivation tray by defining the positioning of each system in the cultivation chamber.
- Improve the quality of the used sensors and actuators to increase the monitoring accuracy and reliability.
- Add some other monitoring capabilities by increasing the variety of used devices (e.g., air quality sensors, heaters or air conditioner control, etc.).

Finally, concerning the control software MVP, the possible future works are:

- Implement an ML algorithm at various levels of the class hierarchy to monitor some parameters (e.g., the temperature variations, the nutrient dispensers, the water consumption, etc.) and to assess the edge computing capability of the system.
- Connect to a cloud service to test the cloud computing compatibility of the developed platform.
- Add a new level of automation by improving the control system classes tree by implementing the VSbc class thanks to the VCsm class to have a finer control over the SBC hardware.
- Develop a proper MD backend server to link the EMS to the MD and bring better control over the growing system.

Chapter 9

Conclusion

This master thesis explored the study, the development, and the testing of a new Automation and Intelligence Platform (AIP) for Smart Plant Factories (SPFs). The literature review focused on the technical Information and Communication Technology (ICT) aspect of the projects showed that a wide variety of implementations exist in the scientific papers, industry reports, and hobbyist works. The hardware and software research has presented a broad availability of suitable hardware components and software frameworks. Moreover, the components and frameworks inter-compatibility was studied alongside the used networking protocols and needed data storage systems (Chapter 4). The development part of the project was properly conducted by implementing a Management Dashboard (MD) frontend and an Electronic Management System (EMS) composed of a control system and many hardware interactions source code. It is pretty substantial and organized seeing that the MD is implemented across 33 files totaling 3016 lines of documented code and that the EMS implementation is scattered over 28 files and 3790 lines of commented code (the detailed information is available in Appendix A). The Minimum Viable Product (MVP) of the control software and the Proof Of Concept (POC) of the monitoring hardware both use a modular approach to be greatly flexible and modifiable (see Chapters 5 and 6). Finally, the AIP was tested thanks to a real world cultivation tray. The performed cultivation experiments proved that the hardware and software side of the implemented AIP work as expected by fulfilling all the wanted requirements (experiments shown in Appendix B).

The hardware and software developments needed by the AIP accommodate most functions required in the deployment of SPFs. Moreover, the developed platform improves upon the currently available solutions thanks to its wide technical documentation, modular design, and powerful hardware and software components. The developed solution does not include advanced automation nor cognitive computing algorithms but was fully thought for accommodating their implementations. The philosophy used during this master thesis was centered around restarting the AIP development from the ground up and thus having a multisectoral approach to homogenize the implementations. This method showed that the final result is more homogeneous and flexible than most of the literature because no other authors did consider the hardware and software developments as a whole.

The concluding remark of this project concerns its future. The implemented AIP is working as expected and already includes some interesting capabilities but presents a wide range of possible future improvements. For each aspect of the development, being the electronic systems, the software implementation, or the cultivation tray integration, many concrete and useful improvements were defined in Chapter 8. This last remark shows the usefulness, the potential, and the interest of such an AIP for transitioning from PFs towards SPFs.

Bibliography

Abdelouahid, R. A., Debauche, O., Mahmoudi, S., Marzak, A., Manneback, P., and Lebeau, F. (2020a). Open phytotron: A new iot device for home gardening. *Orbi, ULiege*.

Abdelouahid, R. A., Debauche, O., Mahmoudi, S., Marzak, A., Manneback, P., and Lebeau, F. (2020b). Smart nest box: Iot based nest monitoring in artificial cavities. *IEEE*.

Araújo, S. O., Peres, R. S., Barata, J., Lidon, F., and Ramalho, J. C. (2021). Characterising the agriculture 4.0 landscape—emerging trends, challenges and opportunities. *Agronomy (Basel)*, vol. 11.

Cabaccan, C. N., Cruz, F. R. G., and Agulto, I. C. (2017). Wireless sensor network for agricultural environment using raspberry pi based sensor nodes. *IEEE*.

Cambra, C., Sendra, S., Lloret, J., and Lacuesta, R. (2018). Smart system for bicarbonate control in irrigation for hydroponic precision farming. *Sensors*, vol. 18.

Chen, W.-L., Lin, Y.-B., Lin, Y.-W., Chen, R., Liao, J.-K., Ng, F.-L., Chan, Y.-Y., Liu, Y.-C., Wang, C.-C., Chiu, C.-H., and Yen, T.-H. (2019). Agritalk: Iot for precision soil farming of turmeric cultivation. *IEEE Internet of Things Journal*, vol. 6.

Chowdhury, M. E. H., Khandakar, A., Ahmed, S., Al-Khuzaei, F., Hamdalla, J., Haque, F., Reaz, M. B. I., Shafei, A. A., and Al-Emadi, N. (2020). Design, construction and testing of iot based automated indoor vertical hydroponics farming test-bed in qatar. *Sensors*, vol. 20.

Debauche, O., Mahmoudi, S., and Guttadauria, A. (2022). A new edge computing architecture for iot and multimedia data management. *Information*, vol. 13.

Debauche, O., mahmoudi, S., mahmoudi, S. A., manneback, p., Bindelle, J., and Lebeau, F. (2020). Edge computing and artificial intelligence for real-time poultry monitoring. *Procedia Computer Science*, vol. 175.

Duston, J. (2017). Assessing the potential environmental impacts of controlled environment agriculture in detroit and the future of this industry based on local food trends. Master's thesis, Harvard University.

Eduardo, C. F., Jake, R., Brander, G., Tim, S., Douglas, C., Hildreth, E., and Caleb, H. (2017). Personal food computer: A new device for controlled-environment agriculture. *CoRR*, vol. abs/1706.05104.

Galanakis, C. M. (2021). *Environment and Climate-Smart Food Production*. Springer International Publishing AG, Chania, Greece.

Jaenuri, A., Hafizh, M., Primatama, S., Nugroho, A. P., and Masithoh, R. E. (2021). Acse (automatic canopy space expander): Automatic spacing system based on increasing plant canopy area on hydroponics for space efficiency. *IOP conference series. Earth and environmental science*, vol. 733.

Kim, T.-H., Lee, S.-H., Oh, M.-M., and Kim, J.-O. (2022). Plant growth prediction based on hierarchical auto-encoder. *2022 International Conference on Electronics, Information, and Communication (ICEIC)*.

Kozai, T. (2007). Propagation, grafting, and transplant production in closed systems with artificial lighting for commercialization in japan. *Propagation of Ornamental Plants*.

Kozai, T. (2018). *Smart Plant factory : The next generation indoor vertical farms*. Springer, Singapore.

Kozai, T., Genhua, N., and Michiko, T. (2019). *Plant factory : An indoor vertical farming system for efficient quality food production*. Academic Press, London, UK.

Kozai, T., Ohyama, K., and Chun, C. (2006). Commercialized closed systems with artificial lighting for plant production. *Acta Horticulturae*.

Kramer, J. (2007). Is abstraction the key to computing? *Commun. ACM*, vol. 50.

Lakhiar, I. A., Gao, J., Syed, T. N., Chandio, F. A., and Buttar, N. A. (2018). Modern plant cultivation technologies in agriculture under controlled environment: a review on aeroponics. *Journal of plant interactions*, vol. 13.

Lin, Y.-B., Lin, Y.-W., Huang, C.-M., Chih, C.-Y., and Lin, P. (2017). Iottalk: A management platform for reconfigurable sensor devices. *IEEE Internet of Things Journal*, vol. 4.

Lin, Y.-B., Lin, Y.-W., Lin, J.-Y., and Hung, H.-N. (2019). Sensortalk: An iot device failure detection and calibration mechanism for smart farming. *Sensors (Basel, Switzerland)*, vol. 19.

Lukito, R. B. and Lukito, C. (2019). Development of iot at hydroponic system using raspberry pi. *Bina Nusantara University, Jakarta, Indonesia*.

Merriam-Webster (2022a). Merriam-webster dictionary: Agriculture. `https://www.merriam-webster.com/dictionary/agriculture`. Last accessed on 2022-04-26.

Merriam-Webster (2022b). Merriam-webster dictionary: Platform. `https://www.merriam-webster.com/dictionary/platform`. Last accessed on 2022-05-18.

Park, S. and Kim, J. (2021). Design and implementation of a hydroponic strawberry monitoring and harvesting timing information supporting system based on nano ai-cloud and iot-edge. *Electronics (Basel)*, vol. 10.

Penghui, X., Nan, F., Na, L., Fengshan, L., Shuqin, Y., and Jifeng, N. (2022). Visual recognition of cherry tomatoes in plant factory based on improved deep instance segmentation. *Elsevier*.

Ralph, E. J. and Brian, F. (1991). Designing reusable classes. *University of Illinois, Urbana-Champaigns*.

Shamshiri, R. R., Kalantari, F., Ting, K., Thorp, K. R., Hameed, I. A., Weltzien, C., Ahmad, D., and Shad, Z. (2018). Advances in greenhouse automation and controlled environment agriculture: A transition to plant factories and urban agriculture. *International journal of agricultural and biological engineering*, vol. 11.

Textier, W. (2013). *Hydroponics for everybody : All about home horticulture*. Mama editions, Paris, France.

Van, L.-D., Lin, Y.-B., Wu, T.-H., Lin, Y.-W., Peng, S.-R., Kao, L.-H., and Chang, C.-H. (2019). Planttalk: A smartphone-based intelligent hydroponic plant box. *MDPI*.

Verma, M. S. and Gawade, S. D. (2021). A machine learning approach for prediction system and analysis of nutrients uptake for better crop growth in the hydroponics system. *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*.

Wildeman, R. (2020). Vertical farming a future perspective or a mere conceptual idea ? a comprehensive life cycle analysis on the environmental impact of a vertical farm compared to rural agriculture in the us. Master's thesis, University of Twente.

Ye, L., Xiaoyuan, M., Lei, S., Gerhard, P. H., and Adnan, M. A.-M. (2021). From industry 4.0 to agriculture 4.0: Current status, enabling technologies, and research challenges. *IEEE transactions on industrial informatics*, vol. 17.

Appendix A

Source Code

This appendix contains all the needed information to get and to analyze the source code presented in this report. The source code repository is available here[1]. All source code is documented and some inline comments are available for algorithms descriptions. The structure of the folder can be seen in the Figures A.1 and A.2 but here is a quick description:

- The folder *EMS* contains all the source code of the Electronic Management System (EMS):
  - The folder *BASE_CLASSES* regroups all the classes composing the control system.
  - The folder *DUMMIES* contains all the dummy objects that can be used to test the system on a development computer (i.e., a computer that has no I/O port for plugging sensors).
  - The other sub-folders of the *EMS* folder regroups all the hardware interactions implementations. These source files are named after the electronic device(s) they are made for and an integer suffix is used for cases where multiple identical devices are used. Alongside these source codes, the needed libraries for some sensors are given and some configuration files can be found.
- The folder *DASHBOARD* contains all the source code of the Management Dashboard (MD)[2]:
  - The folder *mains* regroups all the mains pages of the cross-platform application. The application entry point is the *main.dart* file and all the other Dart files are used by this main file.
  - The folder *components* contains all the source code file implementing a specific component used in the dashboard main pages.
  - The folder *config* regroups the needed configuration files. Three of them are required to configure the dashboard colors, sizes, and responsiveness.
  - The folder *data* contains the mock-up data for the dashboard demonstration screens.
- The file *requirements.txt* contains the list of requirements for each of the previously described folder.

The source code lengths can be seen in Tabular A.1. The general logic of the project implementation was to produce short, structured, and organized source code files. The three implemented systems are detailed: the frontend part of the MD, the control system for the EMS, and the hardware interactions classes of the EMS. Also, to have an honest overview of the code source, Tabular A.2 summarizes the various copyrights: original code, inspired from the web (only the general idea is not original, the whole code is custom), adapted from the web (the source code is similar to the original source).

[1] `https://drive.google.com/drive/folders/15z-d5TlOOKYmwnA3MDQZkPYs2y5oB9Lz?usp=sharing`

[2] Seeing that the dashboard is developed in the Dart programming language with the Flutter framework, only the required files are provided. More precisely, the *DASHBOARD* folder content has to be transferred into the *lib* folder of a fresh Flutter project to be used.

Project	Language	Files	Code	Comment	Blank	Total lines
MD frontend	Dart	33	2422	391	203	3016
EMS control system	Python	7	494	816	97	1407
EMS hardware interactions	Python	21	873	1285	225	2383

Table A.1: Summary of the source code files with detailed line counts.

Origin	Files	Total lines
Original code	50	5636
Inspired from the Web	8	517
Adapted from the web	3	653

Table A.2: Summary of the source code origin with line counts.

As seen in the Tabular A.2, some inspiration for the implementations was taken during the research phase of this master thesis. For copyright purposes, the following listing refers to the original sources. Moreover, only a few functions originate from the web and handful of ideas were adapted from tutorials. When web resources were used during the development phase, the concerned source code files explicitly mention the origin URLs and authors.

- The files *am2315c_library.py* and *am2315c_i2c.py* originate from a GitHub repository[3] and were cleaned up and debugged.
- The files *adc_library.py*, *ec_library.py*, *orp_library.py*, and *ph_library.py* originate from the product pages[4][5][6][7] tutorials and were heavily modified to properly work.
- The structure used in the files *colors.dart*, *responsive.dart*, and *size.dart* was firstly seen in a tutorial[8].
- The widget state logic used in the file *updateTile.dart* originate from a tutorial[9] and its visual appearance was adapted.
- The widget state logic used in the file *graphTile.dart* originate from the Syncfusion package documentation[10] and its visual appearance was adapted.

[3] AM2315C library GitHub repository: `https://github.com/switchdoclabs/SDL_Pi_AM2315`
[4] ADC manufacturer tutorial: `https://wiki.seeedstudio.com/8-Channel_12-Bit_ADC_for_Raspberry_Pi-STM32F030/`
[5] EC manufacturer tutorial: `https://wiki.seeedstudio.com/Grove-EC-Sensor-kit/`
[6] ORP manufacturer tutorial: `https://wiki.seeedstudio.com/Grove-ORP-Sensor-kit/`
[7] pH manufacturer tutorial: `https://wiki.seeedstudio.com/Grove-PH-Sensor-kit/`
[8] Flutter project structure tutorial: `https://morioh.com/p/fd4bce4016b8`
[9] Multi-line Flutter text form: `https://protocoderspoint.com/flutter-multi-line-textformfield-maxline/`
[10] Syncfusion package documentation: `https://pub.dev/documentation/syncfusion_flutter_charts`

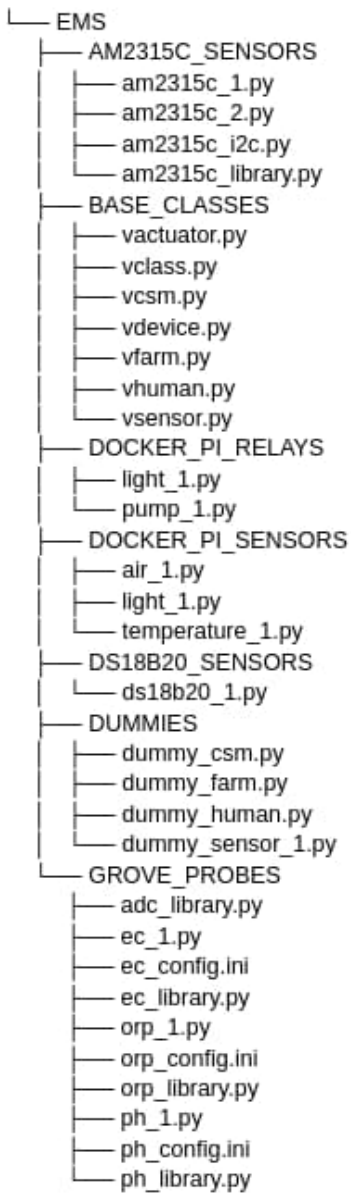

Figure A.1: Tree structure of the *EMS* folder of the source codes repository.

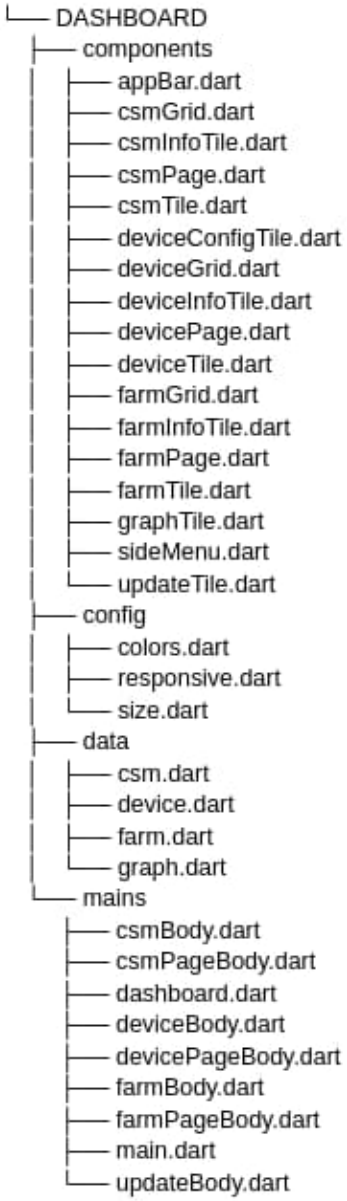

Figure A.2: Tree structure of the *DASHBOARD* folder of the source codes repository.

Appendix B

Cultivation experiments

This appendix gives some insights of the performed cultivation experiments. To give some context, 5 generations of lettuce were grown and 3 of them reached maturity. For the two other experiments, the environmental parameters were not suitable enough for the lettuce to properly grow. Moreover, only the two last growing cycles were conducted with prototypes of the developed AIP. The data from these two experiments are detailed in this appendix.

The charts presented in Figures B.1, B.2, B.3, B.4, and B.5 present the monitored metric during the two last experiments. The first test started on the 16/12/2021 and ended on the 28/02/2022 with fully grown lettuces. The second one started on the 27/03/2022 and ended on the 15/05/2022 with medium lettuces that suffered from leaf scorching. These two tests can easily be seen in the nutrient solution concentration chart (Figure B.2). These 5 charts are presented to show that the EMS and the hardware interactions source code were able to work during long periods of time while monitoring the system status and storing the captured data. Moreover, having these data allows a better management of the system and an improvement of the cultivation environmental parameters.

The most interesting data to analyze is the pH value evolution. This is presented in Figure B.1 and the two performed experiments can be distinguished thanks to the rising pH values around the 27/03/2022. Before and after this, it can be seen that the pH level oscillates around 7,5. This is due to the manual pH adjustment. This is exactly the reason why the advanced design for the electronic system includes peristaltic pumps: to automate the pH adjustments and avoid these oscillations.

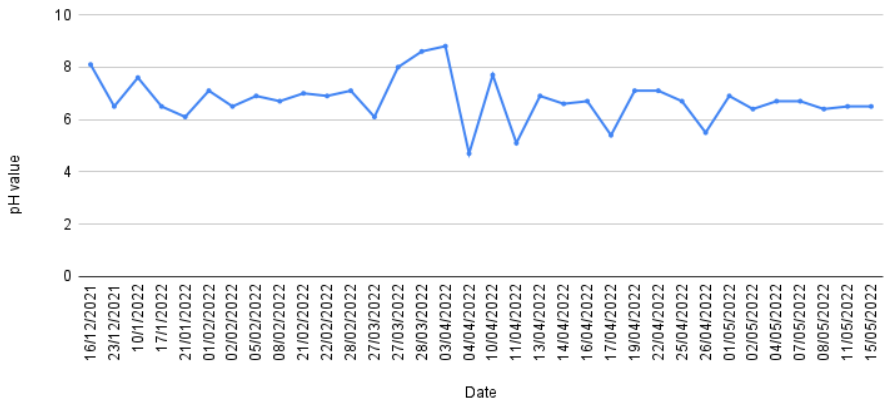

Figure B.1: Nutrient solution pH value evolution during the two last growing experiments.

The nutrient solution concentration evolution presented in Figure B.2 shows the measurements taken by the EC probe. Thanks to the presence of this probe, it can be seen that the concentration level is always increasing. From a cultivation point of view, this trend is not ideal and a given volume of fresh water should have been added once in a while. As it can be observed in this chart, this strategy was adopted in the second experiment. However, thanks to the monitoring system, it can be seen that the added water volumes were not important enough to bring down the concentration level. In fact, when looking at the nutrient solution level in Figure B.3, it can be observed that the decrease of volume is slightly slower during the second experiment but the added volumes are not even perceptible on the data curve.

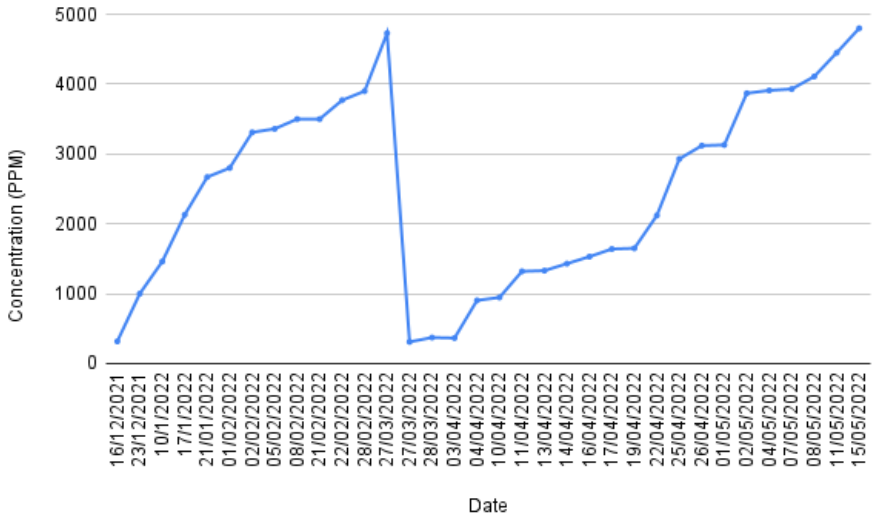

Figure B.2: Nutrient solution concentration (PPM) evolution during the two last growing experiments.

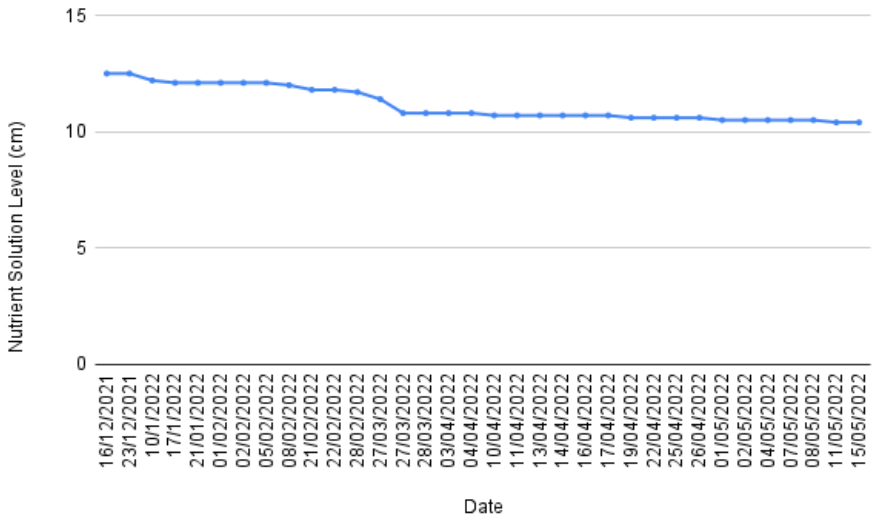

Figure B.3: Nutrient solution level (cm) evolution during the two last growing experiments.

The two final charts are less of interest (Figures B.4 and B.5) because they relate the air and nutrient solution temperature. Seeing that the current system has no control on these temperatures, they are just an indication of the environmental parameters the lettuce were grown in.

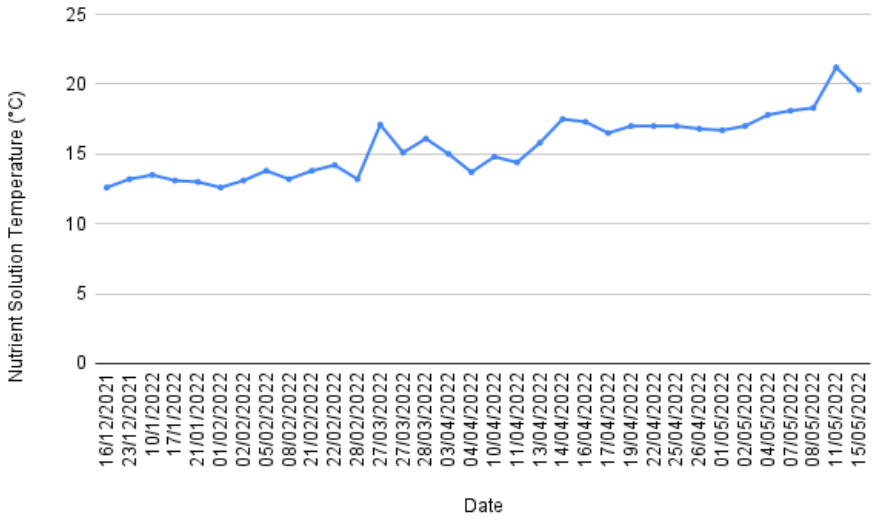

Figure B.4: Nutrient solution temperature evolution during the two last growing experiments.

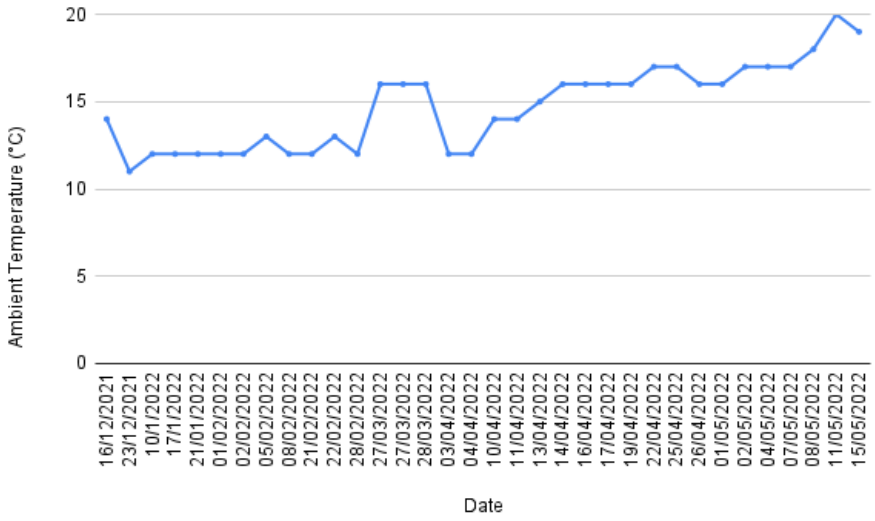

Figure B.5: Ambient air temperature evolution during the two last growing experiments.

Appendix C

Electronic components

Firstly, here is the list of hardware components used to build the electronic assembly Proof Of Concept (POC) of the Automation and Intelligence Platform (AIP) hardware side:

Figure C.1: The AM2315C temperature and humidity sensor (source: product page[1]).

Figure C.2: The DS18B20 waterproof temperature sensor (source: product page[2]).

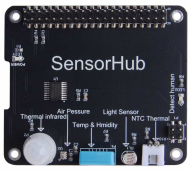

Figure C.3: The EP-0106 HAT (source: product page[3]).

Figure C.4: The EP-0099 HAT (source: product page[4]).

Figure C.5: The 8B-12C-ADC ADC HAT (source: product page[5]).

Figure C.6: The Raspberry Pi Zero 2W (source: product page[6]).

Figure C.7: The pH water quality probe (source: product page[7]).

Figure C.8: The EC water quality probe (source: product page[8]).

Figure C.9: The ORP water quality probe (source: product page[9]).

Figure C.10: The LED panel (source: product page[10]).

[1] AM2315C product page: `https://www.adafruit.com/product/5182`
[2] DS18B20 product page: `https://www.adafruit.com/product/381`
[3] EP-0106 product page: `https://wiki.52pi.com/index.php?title=EP-0106`
[4] EP-0099 product page : `https://wiki.52pi.com/index.php/EP-0099`
[5] 8B-12C-ADC product page: `https://www.seeedstudio.com/8-Channel-12-Bit-ADC-for-Raspberry-Pi-STM32F030.html`
[6] Raspberry Pi Zero 2W product page: `https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/`
[7] pH probe product page: `https://www.seeedstudio.com/Grove-PH-Sensor-Kit-E-201C-Blue-p-4577.html`
[8] EC probe product page: `https://www.seeedstudio.com/Grove-EC-Sensor-Kit-DJS-1C-Black-p-4576.html`
[9] ORP probe product page: `https://www.seeedstudio.com/Grove-ORP-Sensor-Kit-501Z-p-4575.html`
[10] LED panel product page: `https://www.amazon.fr/Bozily-Croissance-Horticole-Minuterie-Fructification/dp/B07K44JKFY/`

Secondly, here is the list of hardware components used to build the advanced design of the electronic assembly of the Automation and Intelligence Platform (AIP) hardware side:

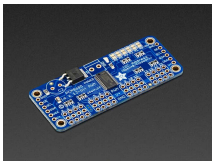

Figure C.11: The PCA9685 I²C PWM driver (source: product page[11]).

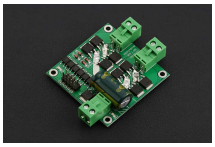

Figure C.12: The DRI0041 DC motor driver (source: product page[12]).

Figure C.13: The LED strip (source: product page[13]).

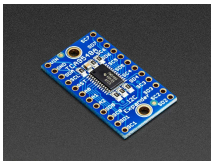

Figure C.14: The TCA9548A I²C multiplexer (source: product page[14]).

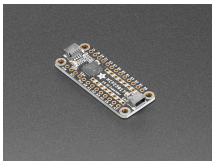

Figure C.15: The MCP23017 I²C GPIO expander (source: product page[15]).

Figure C.16: The DFR0473 relay (source: product page[16]).

Figure C.17: The NF-A8 fan (source: product page[17]).

Figure C.18: The FIT0617 solenoid valve (source: product page[18]).

Figure C.19: The peristaltic pump (source: product page[19]).

Figure C.20: The SEN0368 liquid level sensor (source: product page[20]).

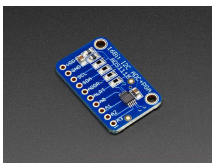

Figure C.21: The ADS1115 16-Bit ADC (source: product page[21]).

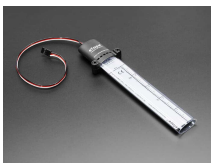

Figure C.22: The eTape liquid level sensor (source: product page[22]).

Figure C.23: The KIT-103P pH probe kit (source: product page[23]).

Figure C.24: The KIT-104O ORP probe kit (source: product page[24]).

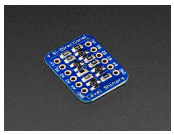

Figure C.25: The BSS138 level I²C safe bidirectional logic converter (source: product page[25]).

[11] PCA9685 product page: `https://www.adafruit.com/product/815`
[12] DRI0041 product page: `https://www.dfrobot.com/product-1534.html`
[13] LED strip product page: `https://www.yujiintl.com/high-cri-95-dynamic-tunable-white-led-flexible-strip/`
[14] TCA9548A product page: `https://www.adafruit.com/product/2717`
[15] MCP23017 product page: `https://www.adafruit.com/product/5346`
[16] DFR0473 product page: `https://www.dfrobot.com/product-1572.html`
[17] NF-A8 product page: `https://noctua.at/en/nf-a8-5v-pwm`
[18] FIT0617 product page: `https://www.dfrobot.com/product-1859.html`
[19] Peristaltic pump product page: `https://www.adafruit.com/product/1150`
[20] SEN0368 product page: `https://www.dfrobot.com/product-2109.html`
[21] ADS1115 product page: `https://www.adafruit.com/product/1085`
[22] eTape product page: `https://www.adafruit.com/product/3828`
[23] KIT-103P product page: `https://atlas-scientific.com/kits/gravity-analog-ph-kit/`
[24] KIT-104O product page: `https://atlas-scientific.com/kits/gravity-analog-orp-kit/`
[25] BSS138 product page: `https://www.adafruit.com/product/757`