UCLouvain

epl

École polytechnique de Louvain

La Plante Connectée

An intuitive low-cost phenotyping station

Author: **Dylan Goffinet**
Supervisor: **Cristel Pelsser**
Readers: **Xavier Draye, Donatien Schimtz**
Academic year 2023–2024
Master [120] in Computer Science

Abstract

The objective of teaching plant physiology is to provide students with an understanding of the biological processes that regulate plant development. It is crucial to offer students practical experience involving direct interaction with developing plants, so that they can observe the processes for themselves using indicators that are visible to the naked eye or measured by sensors. However, the resources and equipment required to enable students to participate actively in the various practical tasks are not always available.

Last year, Louis Lemaire and Martin Lallemand endeavoured to introduce an innovative, interactive methodology to these practical sessions through the development of a low-cost phenotyping system [1]. The system was designed to facilitate the observation and measurement of specific characteristics of an individual plant and to be deployed in multiple units and utilised by groups of students.

The objective of this thesis is to analyse this prototype and to subsequently enhance it with the intention of improving its maintainability, reliability and overall robustness. This should provide a robust foundation for future improvements that could potentially result in the development of a final product that is fit for purpose in practical work.

Acknowledgment

Firstly, I would like to express my gratitude to my supervisor, Cristel Pelsser, for her guidance and support throughout the process of writing this thesis. Her availability, advice, and feedback throughout the year were instrumental in enabling me to make significant progress on this project.

Secondly, I would like to sincerely thank Xavier Draye and Nicolas Biot for their invaluable advice and insights into the various aspects of this work that were outside my area of expertise. Their guidance helped me to better comprehend the scope and objectives of this project.

Furthermore, I would like to thank Donatien Schmitz for agreeing to be a member of my jury and for taking the time to examine my work.

I am also grateful to Pierre Van Thorre for his help with the installation of my experiments in the UCLouvain greenhouses.

Finally, I would like to extend my gratitude to Louis Lemaire and Martin Lallemand, without whom this project would not exist. Their willingness to answer my questions and share their valuable resources was instrumental in the realisation of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of teaching plant physiology is to provide students with an understanding of the biological processes that regulate plant development. It is crucial to offer students practical experience involving direct interaction with developing plants, so that they can observe the processes for themselves using indicators that are visible to the naked eye or measured by sensors. However, the resources and equipment required to enable students to participate actively in the various practical sessions are not always available.

Last year, Louis Lemaire and Martin Lallemand attempted to introduce an innovative, interactive methodology to these practical sessions [1]. To achieve this, they undertook the development of a low-cost phenotyping system, which would facilitate the observation and measurement of specific characteristics of an individual plant, and which could be deployed in multiple units and utilised by groups of students.

However, the possibilities are numerous and it was important to focus first on a limited number of measurements in order to propose a prototype that could serve as a basis for future developments. With this in mind, two indicators were chosen: transpiration and growth of the aerial parts of a plant. Transpiration is measured using a specially constructed balance with a load cell, and aerial growth is measured by analysing images taken with a miniature camera.

The objective of this thesis is to analyse the prototype developed by Louis Lemaire and Martin Lallemand and to subsequently enhance it with the intention of improving its maintainability, reliability and overall robustness. This should provide a robust foundation for future improvements that could potentially result in the development of a final product that is fit for purpose in practical work.

This thesis commences with an overview of plant phenotyping and the existing solutions as well as the technologies employed by the stations. The next chapter will present an analysis of the work undertaken by Louis Lemaire and Martin Lallemand, with a detailed

examination of the proposed prototype and its inherent limitations. The subsequent parts will focus on the various modifications made to the software aspect of the stations, from the choice of operating system to the processing and visualisation of acquired data, and conclude by presenting potential perspectives for improvement identified during the course of writing this report.

Chapter 2

State of the art

This chapter aims to provide a concise introduction to the notion of phenotyping and the existing solutions that have been developed in this field. Furthermore, it introduces some of the technologies that are employed in the phenotyping station proposed by Louis Lemaire and Martin Lallemand, which serves as the basis for the work presented in this thesis [1].

2.1 Phenotyping

Phenotyping is defined as the characterisation of the phenotype. In contrast to genotype, which refers to the expression of the genetic material of an individual or the shared genetic material of a species, the phenotype is the set of observable characteristics of an individual or species. Consequently, it represents the observable traits of an organism, which are dependent on both its genotype and the environmental conditions in which it is situated [2].

In plant physiology, the process of phenotyping refers to the collection and analysis of data on plant characteristics. Plant phenotyping is of particular interest in the variety selection, where it is necessary to know which plants have traits of interest for a given crop. In this context, a non-destructive and high-throughput phenotyping can speed up and improve the efficiency of this selection process [3].

Innovations in phenotyping are making it possible to characterise plant traits with increasing precision and to understand how genetic and environmental factors influence their development. These innovations are part of the drive to improve yields and optimise inputs and production methods.

More generally, plant phenotyping is part of an approach to understanding ecosystems that is essential in the context of sustainable agriculture, as it improves the understanding

of the interactions between plants and their environment.

Phenotyping and plant physiology are of great importance for Bioengineers, as they provide a comprehensive understanding of the various metrics and influencing factors affecting plant development. An in-depth knowledge of these processes is essential for them to comprehend the functioning of a plant and the impact of its environment [4]. However, the process of phenotyping presents a complex challenge with regard to data acquisition. In this context, the utilisation of high-throughput, non-invasive methodologies is essential in order to collect pertinent data whilst exerting a minimal influence upon the development of the plant.

2.1.1 Existing Phenotyping Solutions

Phenotyping methods are being developed and refined to such an extent that they have become a multidisciplinary engineering field in their own right. This has led to the emergence of a number of companies specialising in the design of instrumentation for plant phenotyping [5, 6, 7]. The growing demand for plant phenotyping, particularly under field-scale growing conditions, is driving the development of this equipment for use in precision agriculture and variety selection.

Similarly, a number of research institutes are engaged in the development of high-throughput phenotyping platforms. Examples of such platforms include the *RootPhAir* platform at UCLouvain, which permits the non-invasive study of plant root systems through the application of image analysis techniques [8]. The *Montpelier Plant Phenotyping Platforms* (M3P), which are housed in the Laboratory for Plant Ecophysiology under Environmental Stress (LEPSE), are used for the study of plants' responses to environmental stress [9]. The *Plant Phenotyping for Plant and Micro-organism Interactions* (4PMI) facility, which is attached to the UMR Agroecology of the INRA Dijon, enables the combination of plant material production under controlled conditions with high-throughput phenotypic characterisation [10]. The *Netherlands Plant Eco-phenotyping Centre* (NPEC) is a facility comprising six distinct phenotyping modules, designed to facilitate the observation and analysis of plant growth across a range of spatial scales and levels of environmental control [11].

2.2 Technologies

This section presents the different technologies used in this work. The first, *PlantCV*, is used to process the images taken by the station. The other two, *IndluxDB* and *Grafana*, are two software used for the collection and visualisation of the data collected by the stations.

2.2.1 PlantCV

PlantCV is an open-source Python library developed by the Donald Danforth Plant Science Center with the objective of facilitating plant phenotyping through image analysis [12]. It is based primarily on the popular OpenCV library, which contains a multitude of computer vision and image analysis tools [13]. In the context of this work, PlantCV is utilised for the processing and analysis of the images of the plants captured by the stations. It was determined to be an appropriate choice for this project for a number of reasons.

Firstly, PlantCV is a tool currently employed in a number of research contexts concerned with image processing for the purposes of plant phenotyping. The automated extraction of phenotypic data facilitated by the software increases the efficiency and accuracy of data capture in comparison with manual methods. Additionally, PlantCV is capable of supporting a wide range of image types, including RGB, thermal and near-infrared images, as well as a variety of analysis techniques, including image preprocessing, segmentation algorithms and feature extractions such as colour and shape [14, 15, 16]. This versatility renders PlantCV applicable to a multitude of research projects pertaining to plant phenotyping. Moreover, the open-source nature of PlantCV allows for customisation and integration with other software tools, thus promoting collaborative research and innovation.

Secondly, PlantCV is based on Python, a programming language taught to second-year Bachelor of Bioengineering students at UCLouvain. Thus, students and assistants can understand the steps in the image analysis pipeline and adapt them to suit their own requirements. Furthermore, the fact that PlantCV is based on Python allows for seamless integration into an image acquisition process that is itself written in Python on a Raspberry Pi computer. This facilitates the automation of the image collection and analysis processes, enabling the creation of a coherent and readily comprehensible whole.

In conclusion, PlantCV offers a straightforward approach to fundamental image analysis techniques, directly employing the computational and mathematical principles inherent to image analysis. This encompasses techniques such as thresholding, which consists of converting an image into a binary representation by setting all pixel values above or below a specified threshold to either the maximum (white) or minimum (black) value, thus providing a simplified representation of the image data for subsequent processing.

2.2.2 InfluxDB

InfluxDB is an open-source, high-performance, distributed, and scalable time-series database developed by InfluxData. It is specifically designed to handle large volumes of time-stamped data, which makes it an ideal fit for storing and retrieving data generated by devices connected in a network. In contrast to conventional relational databases, InfluxDB was developed with distinctive indexing and querying capabilities optimised for time-series data [17]. Time-series data, or time-stamped data, is a series of data points arranged in chronological order according to the time at which they were recorded.

InfluxDB offers two different query languages: a custom SQL-like query language, called InfluxQL, and Flux. Both of these languages are designed for working with time-series data. The syntax of Flux has been designed in such a way as to support the construction of chains and pipelines of operations, thereby enabling users to build complex queries that are capable of handling transformations and analytics on data from a variety of sources and across different time ranges. In contrast to SQL, which has been optimised for the joining and relating of normalised data, Flux has been developed with a specific focus on querying, analysing and acting on time series data, thereby offering powerful capabilities for the processing and transformation of data in real time [18].

In addition, the reliability and scalability of InfluxDB makes it a robust choice for this project, as it can accommodate growing data volumes without compromising performance, meaning it can be used with a single station or multiple stations simultaneously. InfluxDB also provides a Python client library [19], which allows for easy integration with the other parts of the software and easy modification if the software needs to be updated to support new measurements, for example.

2.2.3 Grafana

Grafana is an open-source platform that is used for both analytics and the interactive visualisation of data. It enables users to create, explore and share dashboards derived from a plethora of data sets [20]. Furthermore, Grafana is compatible with a broad range of data sources, including InfluxDB. This compatibility is further reinforced by the fact that Grafana supports the Flux query language, which allows for the robust and flexible retrieval and analysis of data [21].

Additionally, Grafana provides a comprehensive array of visualisation options, including charts, graphs, tables and maps, thus affording users the flexibility to represent their data in a manner that aligns with their specific requirements and facilitates comprehension.

This thesis will examine the development and implementation of a phenotyping station that employs PlantCV for the analysis of plant images and the storage of the resulting data in an InfluxDB database, which will be visualised using Grafana.

Chapter 3

PhenoHive: a phenotyping system for students

Last year, Louis Lemaire and Martin Lallemand developed a low-cost single plant phenotyping station as a new innovative approach for the practical sessions of the plant physiology course (LBIR1251) [1].

This chapter will present the motivations behind their proposal, the objectives of the practical sessions and their current organisation. It will also provide an overview of the proposed prototype, its design, and its implementation, which formed the starting point for this thesis.

3.1 Motivation

The aim of teaching plant physiology is to give students an understanding of the biological processes that govern plant development. In this regard, it is essential to offer students practical experience involving direct interaction with developing plants, so that they can observe the processes for themselves using indicators visible to the naked eye or measured by sensors.

However, the resources and materials required to enable students to participate actively in the different practicals are not always available. Indeed, the practical sessions require sensors for each plant, such as a precision balance with a data-logger to measure the metrics required. These sensors are often not available in sufficient quantity to allow each group of students to interact with them. This presents a particular challenge for courses followed by more than a hundred students each year, as is the case with the plant physiology course.

In order to meet the requirements of the practical work, the decision was taken to develop a phenotyping system that would facilitate the observation and measurement of specific characteristics of an individual plant. Furthermore, the system was to be low-cost and deployable in multiple units to enable its use by groups of students. Additionally, the design aimed to serve as a foundation for future enhancements, such as the integration of a new sensor, potentially extending the system's utility and enabling the measurement of more relevant metrics pertinent to the practical work.

3.2 Practical sessions: organisation and objectives

The plant physiology course (LBIR1251), which counts for 5 ECTS, is taught by Xavier Draye and Stanley Lutts, and the practical sessions are supervised by Lola Leveaux and Nicolas Biot. The course is organised by the Faculty of Bioengineering at UCLouvain, in the second term of the second year of the Bachelor in Bioengineering. It is composed of 30 hours of lectures and 30 hours of practicals, held in auditoriums and greenhouses [22].

Students are divided into a number of groups according to the number of students enrolled on the course. In the academic year 2023-2024, 170 students were divided into eight groups.

Each group participates in eight practicals distributed across the second term, from week 2 to week 12. These are organized into five thematic blocks, comprising a total of eight sessions. Three of these sessions are conducted in the greenhouses, while the remaining five are held in the auditorium, as illustrated in Table 3.1 [4].

Weeks	Thematic blocks	Sessions
Week 2 or 3 Week 4	Water	A session in greenhouses (manipulation) A session in auditorium (analysis)
Week 5 or 6	Mineral nutrition	A session in greenhouses (manipulation)
Week 7 or 8 Week 10	Photosynthesis	A session in greenhouses (manipulation) A session in auditorium (analysis)
Week 9 and 11	Creating an experiment protocol	Two sessions in auditorium
Week 12	The environment	An experiment to be carried out at home throughout the term A session in auditorium (analysis)

Table 3.1: Schedule and session types for the five thematic blocks of the practicals in LBIR1251 [4]

These topics are organised in such a way as to provide students with a practical understanding of the key processes that govern how plant function, as well as the best

practices to follow when setting up a plant physiology experiment. Furthermore, these labs allow students to develop skills that extend beyond plant physiology, such as the ability to store and present data in graphical form to facilitate the interpretation of results. Additionally, students learn to discuss and debate in groups, as well as develop a critical view of experimental results [22, 4].

It is therefore imperative that the system developed enables the attainment of the greatest number of learning objectives in terms of knowledge and skills, while facilitating a more interactive approach to the practical sessions.

Nevertheless, it was also crucial to initially restrict the scope of the station, as a plethora of sensors can be employed to monitor a multitude of variables, thereby creating a vast array of potential configurations for the station.

Several possibilities were initially considered, such as measuring photosynthesis using CO_2 sensors, analysing fluorescence or studying plant growth as a function of exposure to light of different wavelengths. However, these were quickly abandoned in favour of aerial growth and transpiration, which initially appeared more straightforward to implement and more interesting to observe in practical sessions.

3.3 Methods and materials

The system can be divided into two parts, which will be detailed below. The first part concerns transpiration monitoring, while the second part is concerned with characterising the aerial growth of the plant. Transpiration analysis is conducted through the measurement of the mass of a potted plant using a scale device. Aerial growth is measured through the image analysis of the plant, which is captured with a camera to extract its height.

3.3.1 Transpiration analysis

An understanding of the processes involved in the movement of water within plants is essential for effective management of crops and the identification of varieties with greater resilience to drought conditions. The movement of water within a plant ecosystem has a significant impact on the yield of cultivated plants, making it a crucial aspect of plant phenotyping. It is therefore essential for Bioengineers to possess a comprehensive knowledge of this field of plant physiology.

Evapotranspiration, defined as the combination of transpiration and evaporation, is of particular significance. Transpiration refers to the process by which water is lost from a plant through its stomata[1]. It plays a critical role in the transport of nutrients, photosynthesis, growth, and the maintenance of internal organism homeostasis [23].

[1]A stoma is a pore or aperture surrounded by two guard cells that allow gas exchange.

One of the most widely employed methods for quantifying plant transpiration is the measurement of the mass of a potted plant. The variation in mass of a potted plant is primarily attributable to transpiration by the plant and the evaporation of water from the soil. To a lesser extent, this is influenced by the gain in dry matter. Consequently, it is feasible to ascertain the transpiration of a potted plant over time by deducting the mass of a control pot devoid of a plant. This is an indirect approach, yet it is straightforward, dependable and non-intrusive.

In order to enable the phenotyping station to collect plant weight, a single-point load cell was coupled with an analogue-to-digital converter. These components are characterised by the fact that they are inexpensive yet programmable using well-known languages such as Python, MicroPython and Arduino, which facilitates the integration of the device within the other parts of the system [24, 25].

A prototype of the balance, illustrated in Figure 3.1, was modelled by Louis Lemaire and Martin Lallemand and 3D printed. The dimensions of the prototype have been designed with the specific intention of accommodating the TAL226 load cell[26], while simultaneously allowing for the housing of the HX711 converter[27]. The base and load plate have been designed as separate components, with a small gap between them to permit the plate to support the maximum load of the load cell (3 kg) without contact with the base. Edges have been incorporated to enhance the rigidity of the device and to safeguard the components from the external environment.

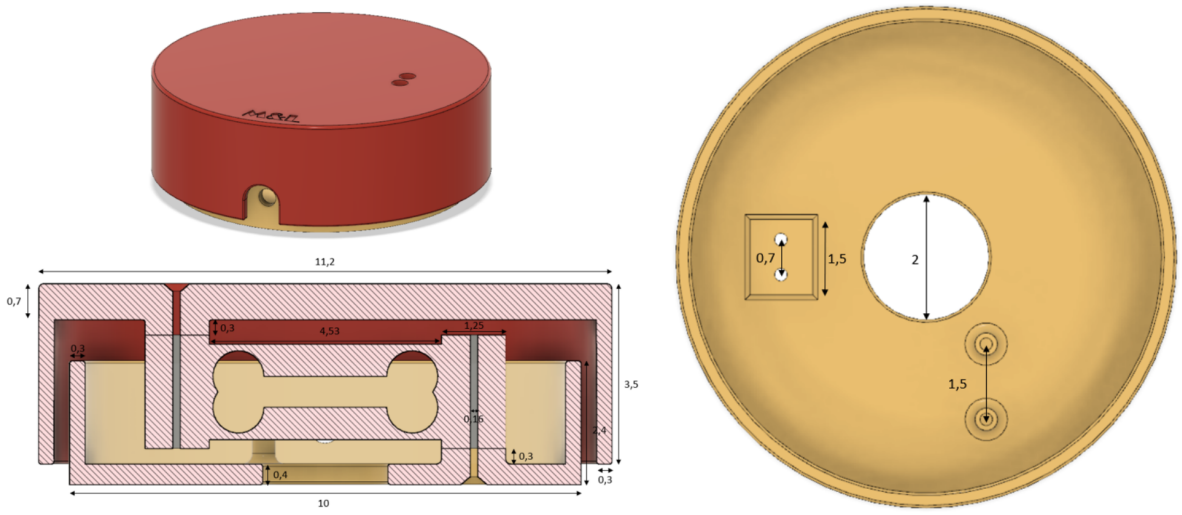

Figure 3.1: Model of the scale with dimensions (cm). External view of the device (top left), sectional view (bottom left) and lower part of the scale (right) [1].

3.3.2 Aerial growth characterisation using image analysis

The growth of the above-ground parts of a plant is an important indicator of its development and is closely linked to several biological processes. Factors such as photosynthesis rate, transpiration rate and nutrient assimilation influence this growth. Monitoring the growth of aerial parts is crucial for practical work, as it helps students to understand and analyse these fundamental processes. By taking frequent measurements of above-ground growth, we can observe the effects of environmental factors such as temperature, sunlight and circadian rhythms on plant development. These frequent measurements also allow comparisons to be made with other physiological processes in the plant, such as transpiration.

In the field of plant phenotyping, image analysis offers a number of advantages that are specific to remote sensing measurements, that is to say, measurements made with no direct contact between the instrument and the object of interest. For instance, image acquisition is non-invasive and does not affect the development of the plant, thus avoiding any potential systematic biases associated with the handling of the plants during the measurement [3, 28]. Similarly, the images are not affected by the experimental effect, provided that the acquisition and analysis of the images is automatic, thereby reducing the influence of the experimenter. This is important if observations of plant growth are to be made in the best possible conditions, particularly in a greenhouse where space is limited and several dozen students have to pass through in a short space of time.

One of the benefits of using image analysis in practical sessions is the versatility of the equipment employed. By utilising a single camera sensitive to the visible spectrum, it is possible to ascertain the leaf area of a plant, the dimensions of its organs and the number of leaves, in addition to analysing the colours present. The camera can be positioned at different angles without affecting the data acquisition process; however, once it is in place, it should not be modified. Consequently, if the objectives of practical sessions change over time, it is not necessary to change the equipment to carry out a relevant analysis. Only the placement of the camera and the analysis of the images need to be reviewed.

The hardware employed for image capture is the Raspberry Pi Camera, which possesses the advantages of being both inexpensive and compatible with Raspberry Pi hardware, in addition to being controllable via the Picamera 2 Python library [29]. In order to ensure sufficient illumination for the imaging of the plant, an LED light strip was incorporated into the design to provide an adequate level of light.

The analysis of images is conducted using the PlantCV Python library, which offers a clear and accessible approach to employing fundamental image analysis techniques such as thresholding, as discussed in section 2.2.1.

3.3.3 Selection of suitable plant species for analysis

In their thesis, Martin Lallemand and Louis Lemaire considered two plant species for analysis by the station: *Arabidopsis thaliana* and *Zea Mays*. The following criteria were required:

Firstly, the plant should demonstrate sufficient transpiration to enable the observation of significant variations in transpiration between day and night using a balance. Secondly, it should be suitable for image analysis and for measuring the growth of its aerial parts with a minimum of spatial fluctuation over time. Thirdly, it should be resistant to frequent handling, given that it will be used by students during practical sessions. Thus, it is important to select a plant capable of withstanding the various stresses induced by these sessions.

Following a series of experiments, *Zea mays* was identified as a suitable species for automated analysis. Firstly, *Zea mays* exhibits relatively stable height growth over time, which allows for the establishment of a straightforward and reliable image analysis pipeline based on the measurement of plant height. Secondly, daily variation in transpiration was easily discernible using precision balances. Given the precision of the scale developed in this project, it is possible to measure this variation using the device.

Image analysis using PlantCV

In order to analyse the aerial growth of the plant, an image pipeline was implemented [1]. This pipeline comprises several steps, which are described in detail below and illustrated in Figure 3.2.

Figure 3.2: Image analysis pipeline using PlantCV

Firstly, an image of the plant is captured from its side. It is imperative that the entirety of the plant is visible within the field of view. Furthermore, the image should be taken in front of a uniform background in order to reduce noise as much as possible.

Once this image as been captured, it is converted to a greyscale based on the black (k) channel of the CMYK[2] colour space to create a mask of the plant.

Then, a Canny edge detection is performed on the greyscale image to detect the edges of the plant in the image. The *Canny edge detection* is a method used for detecting contours in an image, developed by John F. Canny in 1986 [30]. It involves applying convolution masks to the image in order to calculate a gradient at each point. Subsequently, thresholding is applied to the gradient map in order to obtain a binary image of the contours. Prior to edge detection, a Gaussian filter with a standard deviation of σ pixels is applied to the image. This step adds blurring to the image, with the aim of preventing small irrelevant elements from being detected as contours.

In order to monitor the plant's growth, it is necessary to extract its structural characteristics from its aerial parts. However, the skeleton can only be obtained from a binary image that includes the full shape of the plant. The Canny filter, however, only renders the edges of the plant, and thus these edges need to be filled. To do so, a morphological closure was performed, which involves dilating an image and then eroding the dilated image using the same structuring element. As a result, the holes and gaps are filled while the overall shape and size of the plant is preserved.

Once the plant's skeleton has been obtained, PlantCV can be used to calculate the sum of the lengths of the plant's segments by summing the amount of pixels that comprise them, and therefore to calculate its growth.

3.4 Initial prototype

This section will provide an overview of the hardware components, software implementation, and usage of a PhenoHive station as proposed by Martin Lallemand and Louis Lemaire. It will conclude with a discussion of the limitations identified in the provided prototypes.

3.4.1 Hardware

Table 3.2 presents a list of the components required to assemble a station, together with the latest prices found online at the time of writing this report. Details of these prices can be found in Appendix B.

[2]CMYK stands for Cyan, Magenta, Yellow and Key (black)

Component	Quantity	Price (€)
Raspberry Pi Zero W with headers	1	19.79
Micro USB Power Supply (5V, 3A)	1	2.71
ST7735 LCD screen (128*160)	1	1.90
Button	2	0.19
LED lighting strips	1	1.69
Relay module KY-019	1	0.56
Picamera with Raspberry Pi Zero adapter	1	2.86
Tal226 load cell	1	11.54
HX711 converter	1	1.17
16 GB SD card	1	3.24
WAGO splicing connector with lever (5 connectors)	3	3.3
Scale device (3D printing)	1	5
Cable (4 wires) 1.5m	1	1.37
Female to female cable	10–15	0.6
Male to female cable	10–15	0.6
Male to male cable	10–15	0.6
TOTAL		57.12

Table 3.2: Hardware components and pricing for building a PhenoHive station

As shown on the last row, the cost of the station is approximately 58 euros, which fulfils the objective of providing a low-cost phenotyping station. It is also noteworthy that the prices listed above represent some of the lowest prices found online for individual components. However, it is possible that some components may be available at a lower cost if purchased in bulk. Furthermore, the cost to print the scale device is an estimated price based on the amount of filament used for printing (approximately 250 grams) and does not include the costs associated with the printing machine.

Figure 3.3 presents a labelled open view of the station. Appendix C contains a guide that details the appropriate wiring of the various components, accompanied by a schematic representation of the station's circuitry.

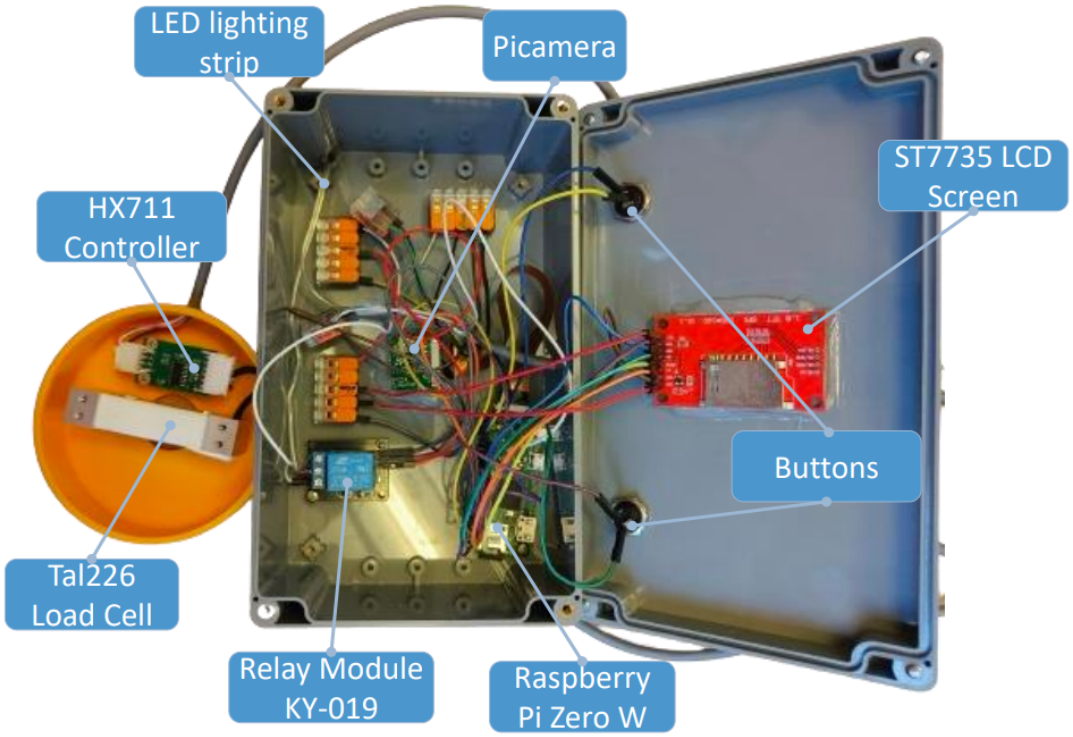

Figure 3.3: Labelled open view of a PhenoHive station

The central component of the station is a Raspberry Pi Zero W, a popular microcomputer that is simple to use and supported by various Linux distributions, including the Raspberry Pi OS, on which a Python script can be readily configured and executed. Additionally, it is compatible with the Picamera and features headers that facilitate direct wiring of the various components. However, it is less powerful than other Raspberry Pi versions, but is sufficient for this project.

The ST7735 LCD screen displays the various menus of the software, as detailed in section 3.4.2 below. The buttons enable the user to interact and navigate through these menus.

As previously discussed in section 3.3.2, the LED lighting strip is connected to the relay module KY-019 to guarantee a consistent light source, even at night, with the objective of enhancing the quality of the images captured by the Raspberry Pi Camera.

The scale device, comprising a Tal226 load cell and an HX711 controller, is visible on the left of Figure 3.3. The HX711 controller is connected to the other components via a 1.5-metre cable. It is essential that the cable length is sufficiently long, as the station must be situated at a certain distance from the plant in order to enable the entire plant to be captured by the camera.

3.4.2 Software configuration

Network configuration

In order to operate, the stations must be connected to an access point (WiFi network) to which the InfluxDB database is also connected. This connection enables the data collected by the various stations to be sent to the database. The data can then be accessed via the Grafana software interface. While it is not mandatory for the access point to be the device containing the InfluxDB and Grafana software, this approach facilitates the configuration of the stations and restricts the devices that can connect to the network. Appendix D provides a guide to configuring a Raspberry as an access point, as well as installing and configuring InfluxDB and Grafana.

Once the access point has been configured, the station can be connected to the network. Figure 3.4 shows the architecture of the PhenoHive network, with InfluxDB and Grafana installed on the access point.

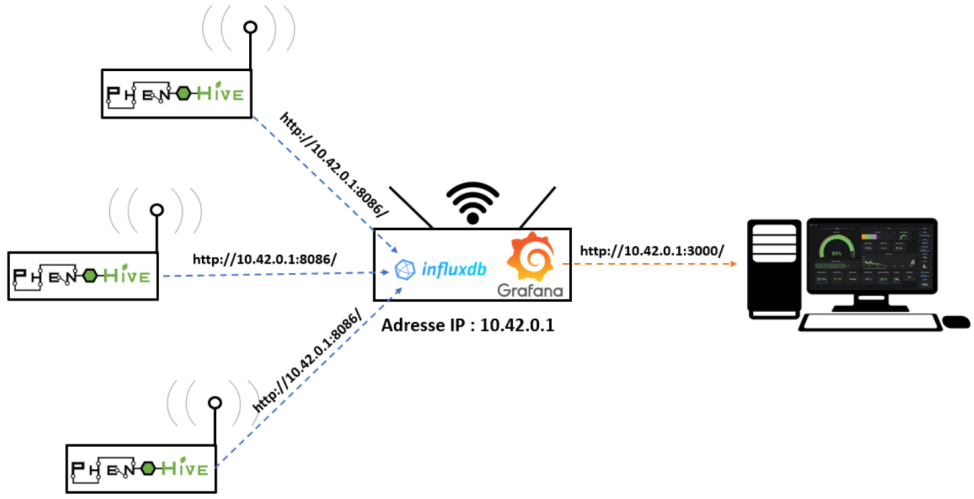

Figure 3.4: PhenoHive network architecture [1]

As illustrated, each PhenoHive station is connected to the same central device via its WiFi network. The benefit of having all the devices on the same wireless network is that they can communicate with each other via their Internet Protocol (IP) address. In this instance, the stations transmit their data to the IP address of the access point and host of the database (`10.42.0.1`).

In other words, an IP address on a network serves to identify the devices that are connected to that same network. Each IP address is associated with a series of ports, which facilitate communication with the device's various software applications. The InfluxDB software is installed by default on port `8086`, while the Grafana software is installed on port

3000. Both InfluxDB and Grafana can be accessed using the Hypertext Transfer Protocol (HTTP) from any browser by entering the corresponding url: `http://HOST_IP:8086/` for InfluxDB and `http://HOST_IP:3000/` for Grafana.

Station configuration

The initial operating system deployed on the stations is the Raspberry Pi OS Desktop 32 bits, which is a Linux distribution optimised for Raspberry Pi hardware and features a graphical desktop interface.

Once the operating system has been installed on the SD card, it can be inserted into the Raspberry Pi Zero W. The user can then download the software from the GitHub repository, install the necessary packages and set up the system to launch the software. These installation steps will be discussed in greater detail in section 5.2.

Once the software has been installed, it needs to be configured. To do so, the configuration file `config.ini` needs to be modified to fit the new configuration. The content of the file is the following:

```
[ID_station]
id = 1

[InfluxDB]
token = TOKEN
org = pheno
bucket = phenostation_data
url = http://10.42.0.1:8086

[Path_to_save_img]
absolute_path = /home/pi/Desktop/phenostation/assets

[image_arg]
pot_limit = 0
channel = k
kernel_size = 3
fill_size = 1

[time_interval]
time_interval = 60

[Var_Verif]
is_shutdown = 0

[cal_coef]
```

```
load_cell_cal = 1
tare = 0
```

Several parameters need to be modified for each station:

- The `ID` of the station serves as the unique identification number for each station. It is used to distinguish the different stations in the database.
- The variables related to the InfluxDB database to fit the configuration specified in appendix D.
- The `absolute_path` parameter should not be modified in the majority of cases, assuming that the repository has been installed on the Raspberry Pi desktop. Its purpose is to locate the various images in the assets folder and to save certain images for subsequent image analysis.
- The parameters associated with `image_arg` are employed for the configuration of the filters utilized for the analysis of the images captured by the station. The `pot_limit` parameter represents the height in pixels above which the pot is no longer visible. The image area falling below the specified threshold is excluded from the subsequent analysis. The `channel` parameter defines the channel to be used for transforming the image into shades of grey. By default, the K channel of the $CMYK$ colour system is used. The `kernel_size` defines the size of the kernel used for the Canny filter. The `fill_size` is used to reduce noise after skeletonisation by filling objects whose size (in pixels) is smaller than this value.
- The `is_shutdown` parameter is used by the station to know if the measurement phase as been interrupted or not; it should not be modified.
- Similarly, the parameters associated with `cal_coef` should not be modified. They are utilised to store the tare and calibration coefficient as it was measurement using the calibration menu.

3.4.3 Using the graphical interface

Once the station has been constructed, and the software has been configured, it can be started. The following section will describe the functioning of the graphical interface of the station as it was received in September.

Main menu

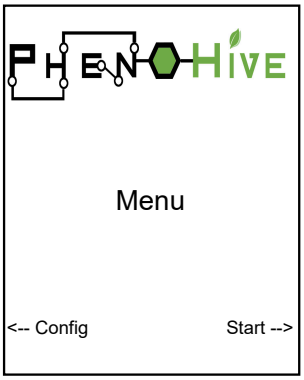

Figure 3.5: Main menu

Once the software has been initialised, a menu will be displayed, as shown in Figure 3.5. This menu enables the user to navigate to the configuration menu by pressing the button on the left, or to begin the measurement phase by pressing the button on the right.

Configuration menu

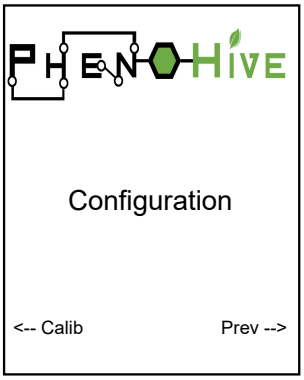

Figure 3.6: Configuration menu

Figure 3.6 shows the configuration menu, which allows the user to navigate the calibration menu using the button on the left and to enter preview mode using the button on the right.

Calibration menu The purpose of this menu is to calibrate the scale. Upon entering the menu, the software performs a tare measurement; therefore, it is imperative to ensure that there are no objects on the scale before entering the menu.

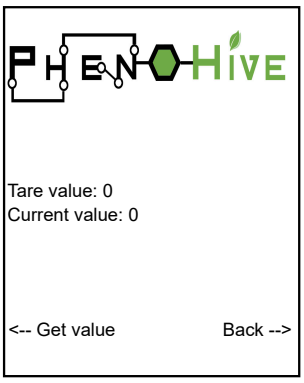

Figure 3.7: Calibration menu

As illustrated by Figure 3.7, the tare value is displayed once inside the menu. The user should then place a weight of 1.5 kilogram to calibrate the scale and press the '`Get Calib`' button on the left. The button should be pressed multiple times to ensure that the value taken into account is not abnormal. The calibration is stored in the station, thus the scale does not need to be recalibrated each time the station is restarted.

Preview mode Upon entering preview mode, the camera will be activated and the software will display the image captured. This facilitates the correct positioning of the station, ensuring that only the plant is visible within the field of view of the camera, as illustrated by Figure 3.8.

Figure 3.8: Preview mode

The preview mode can be exited at any moment by pressing the button on the right.

Measurement menu

Once the station as been configured, the measurement phase can be initiated from the main menu by selection the `Start` option.

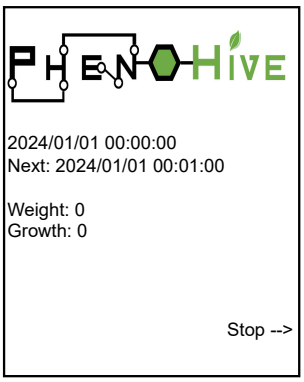

Figure 3.9: Measurement menu

As illustrated by Figure 3.9, the menu displays the current date and time, as well as the time at which the next measurement will be taken. The interval between each measurement is 60 seconds by default, but can be modified in the configuration file `config.ini`. The data measured during the previous measurement by the scale and the image analysis are also displayed. The measurement phase may be interrupted by pressing the button on the right, which will navigate back to the main menu.

Data acquisition mode Once the scheduled measurement time is reached, the station will automatically begin the data acquisition process. Figure 3.10 illustrates the display that will be visible during this period. Once the measurement has been completed and transmitted to the database, the station will return to the measurement menu.

Figure 3.10: Data acquisition mode

3.4.4 Limitations

Four prototypes were constructed and evaluated by Louis Lemaire and Martin Lallemand during the previous year. While the results were promising, their design and software

implementation are not optimal and introduce several limitations.

Firstly, the Raspberry Pi Zero W has a restricted hardware capacity, which results in a relatively slow operational speed. This is further accentuated by the operating system installed on the device. While the Raspberry Pi OS Desktop offers a comprehensive desktop environment, enabling the connection of a keyboard, mouse, and screen for ease of interaction with the system, it is a rather demanding operating system for the limited hardware capabilities of the device. Indeed, the time required for a station to boot can reach several minutes, due to the demanding nature of the operating system and the constraints imposed by the hardware's limited capacity.

Secondly, the software developed for the stations, although operational, is not sufficiently robust and susceptible to crashes. This is problematic as the software is set up to reboot in the event of a crash and does not feature any logging system. Consequently, the user may not be aware that the station encountered an error nor what this error was. Furthermore, the software is divided into only three Python scripts with very few accompanying documentation, which renders any modification both time-consuming and challenging.

Thirdly, the weight collection using the Tal226 load cell and the HX711 is not optimal and frequently yields anomalous readings, as illustrated in Figure 3.11. The source of these aberrant values is not known; however, they may be attributable to interference from nearby devices or noise in the circuitry.

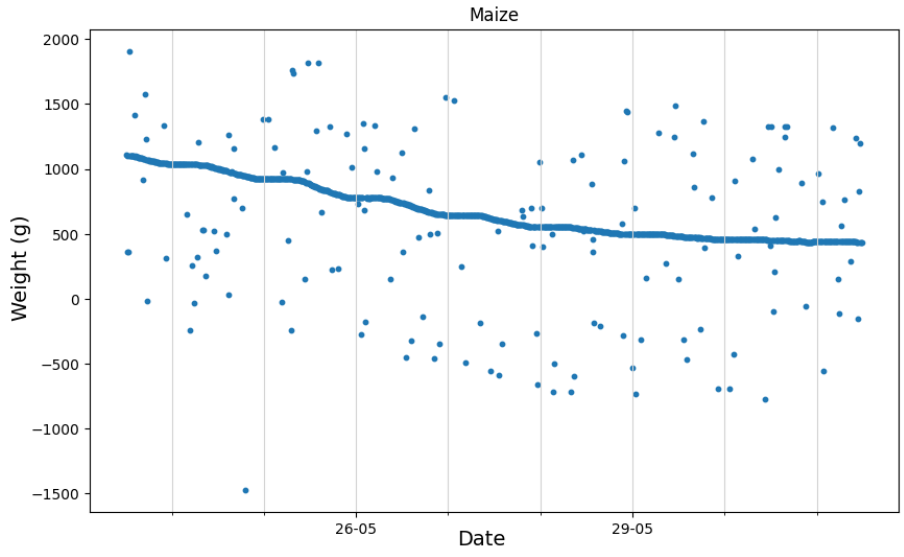

Figure 3.11: Anomalous data collected by the scale

Lastly, the image analysis requires that the station be correctly positioned, ensuring that only the plant is visible within the camera's field of view. This encompasses the plant pot

and any shadows cast over the background, which could potentially be detected as edges and result in erroneous image processing. Figure 3.12 illustrates how a shadow can impact the image processing.

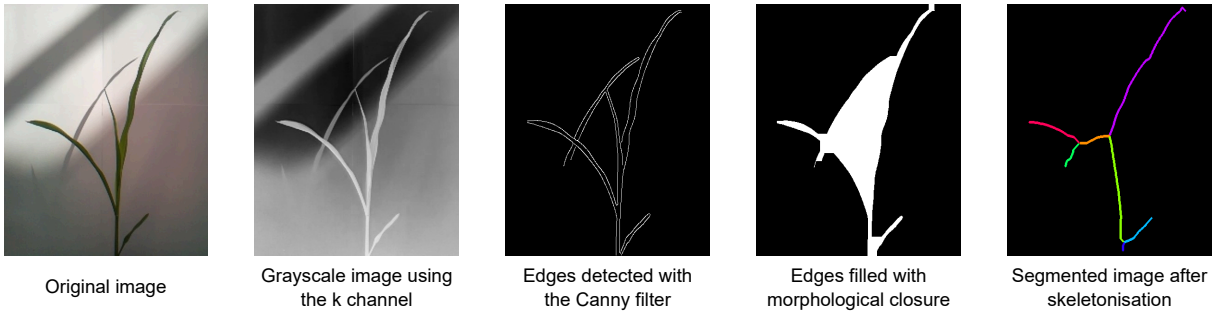

Figure 3.12: Example of an erroneous image processing due to a shadow

The following chapters will describe the various modifications made to the software aspect of the station with the intention of addressing the aforementioned limitations and establishing a more resilient foundation for future developments.

Chapter 4

Operating System

This chapter will examine the reasoning behind the selection of a new operating system, the considerations that led to this choice, and the alternatives that were considered. It will conclude with an explanation of the motivations behind the final decision.

4.1 Limitation of the current distribution

The prototype proposed by Louis Lemaire and Martin Lallemand is based on the Raspberry Pi OS Desktop 32 bits. Raspberry Pi OS is a free, Unix-like operating system based on Debian Linux and optimised for use with Raspberry Pi hardware. Nevertheless, despite the optimisation of these distributions for Raspberry Pi hardware, the Raspberry Pi model utilised in our system is the Raspberry Pi Zero W, which is equipped with constrained hardware capacity.

Indeed, the Raspberry Pi Zero W is equipped with a relatively simple 1GHz single-core ARMv6 CPU (BCM2835) and 512MB of RAM. The limited processing power and memory capacity of the device can result in performance issues when running a full desktop environment. Furthermore, the base version of Raspberry Pi OS Desktop comes preinstalled with several programs that may not be necessary for our specific use case, in addition to the running desktop environment This results in a waste of space and resources. To this end, a decision was made to replace the distribution currently in use for our system with a more lightweight alternative. A comparison of the different distributions tested is presented in the following section.

4.2 Comparison between different lightweight distributions

A multitude of Linux distributions are available, ranging from those with an ultra-minimalist configuration, such as Tiny Core Linux, to those offering a comprehensive desktop environment, such as Ubuntu. However, we can easily reduce the scope of available distribution to only consider the ones relevant to our system. To do so, we considered the following criteria:

- Only distributions that have a stable release.
- Lightweight distributions that have good support for Raspberry Pi hardware.
- Ease-of use and decent community size.

First, choosing a distribution with a stable release guarantees reliability and reduces the likelihood of bugs or instability. Stable releases have been thoroughly tested by the maintainer and the community, and are usually more secure and reliable.

Secondly, using a lightweight distribution optimised to use minimal system resources is crucial given the limited resources of a Raspberry Pi Zero W. Furthermore, good hardware support ensures that Raspberry Pi features such as GPIO pins and PiCamera support, etc. are available without the need to install additional drivers and extensive manual configuration.

Lastly, the ease of use and accessibility are important to be able to streamline the configuration and installation process of new stations without requiring extensive knowledge of the Linux operating system. Additionally, with a reasonable community size, it is easier to find troubleshooting tips and support, reducing the time required to find solutions to any distribution-specific problems that may arise.

Following these requirements, four additional distribution were selected:

- Raspberry Pi OS Lite: a minimal version without graphical interface of the currently used Raspberry Pi OS [31].
- DietPi: an optimised and lightweight Debian distribution designed for minimum resource usage and maximum performance [32].
- Alpine Linux: a popular lightweight and security-oriented Linux distribution [33].
- piCore: a minimalist variant of Tiny Core Linux for Raspberry Pi, running entirely in RAM [34].

Table 4.1, available at the end of this section, presents a comparison of various metrics for each distribution, with the first column being the Raspberry Pi OS Desktop distribution

that has been used until now, for comparison purposes. The commands employed to gather each metric are provided at the end of this section, in Table 4.2. All metrics were obtained following a new installation of the corresponding distribution on a Raspberry Pi Zero W with a 16 GB SD card. The different metrics used are the following:

- The release date indicates when the version used in the tests was released.
- The base Linux distribution indicates the underlying Linux distribution on which the current distribution is based. It serves as an indicator of the potential compatibility with common software. For example, distributions based on Debian, such as Raspberry Pi OS and DietPi, will have access to the popular Debian package repositories.
- The Linux kernel version shows the version of the Linux kernel used by the distribution, which can affect hardware compatibility, performance, security features, etc.
- The image size shows the size of the distribution's image file that needs to be downloaded and written to the SD card.
- The disk usage indicates the amount of disk space used by the distribution after a fresh installation, showing how much space is used just by the operating system.
- The Random Access Memory (RAM) usage shows the amount of memory used by the distribution after a fresh installation and with no processes running other than those installed with the distribution.
- Preinstalled packages indicated the number of software packages that are installed with the distribution. A higher number of preinstalled packages can indicate a more complete ready-to-use experience, but may also mean a larger image size and higher resource usage.
- The boot duration is the time it takes for the distribution to boot from power-on to a usable state. A faster boot will mean that the station will be up and running faster after a reboot or a crash, for example.
- Running processes shows the number of processes running on the system after booting, on a fresh installation of the distribution. Fewer running processes may indicate lower resource usage, leading to better performance and stability.

As we can see, all the distributions tested are recent, the oldest being piCore, which was released on 17 April 2024. Furthermore, all the distributions use Linux kernel version 6.1 or 6.6, both of which are Long-Term Support (LTS) versions and for which security support is scheduled to end on 31 December 2026 [35].

Figure 4.1 presents a graphical comparison of the remaining metrics gathered, namely those related to the resource usage of each distribution.

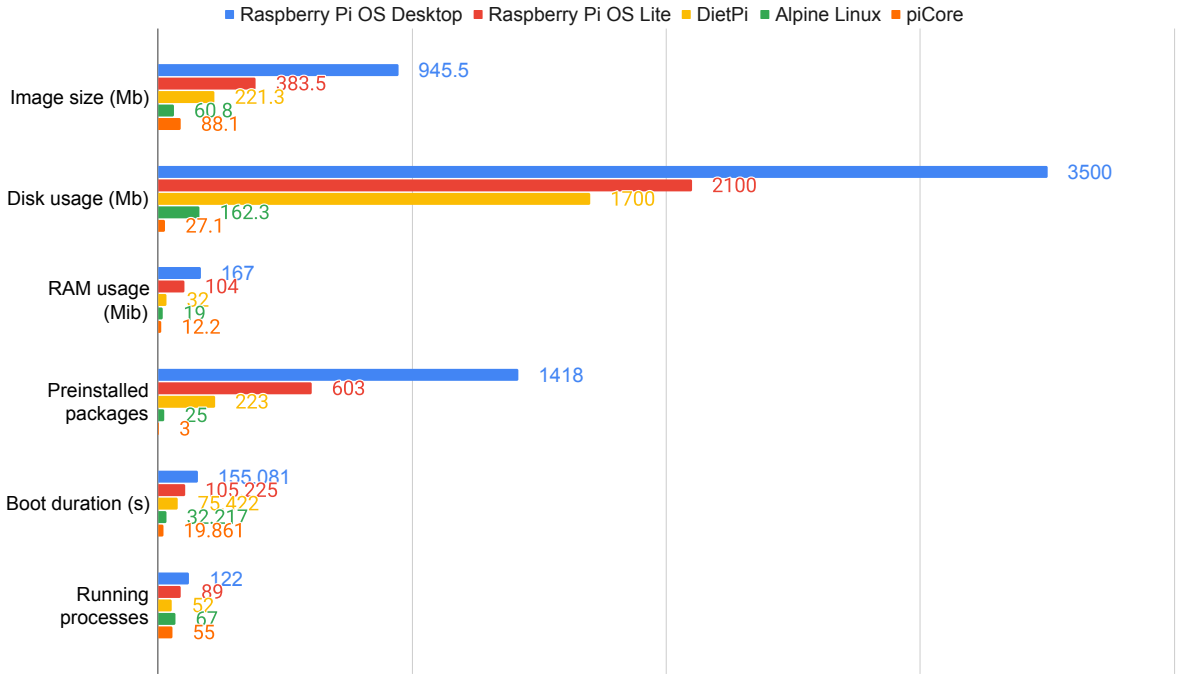

Figure 4.1: Comparison between the different resource usage metrics of each distribution.

In terms of image size and disk usage, there is a big difference between Alpine and piCore, which have very small images and use little disk space, and the other distributions. This can be explained by the fact that they are RAM-only distributions and come with a very limited set of preinstalled packages, as we will see later. However, it should be noted that DietPi is also relatively small, while Raspberry Pi OS Lite is the largest of the four, but is still light compared to a full desktop distribution.

Looking at RAM usage, we can see a clear gap between Raspberry Pi OS Lite and the other distribution, which uses significantly less memory. This is crucial for our hardware, which only has 512Mb of RAM available. Similarly, we can see that Raspberry Pi OS Lite comes with a relatively high number of preinstalled packages, which can provide a more complete out-of-the-box experience, but can also explain the gap between the amount of RAM used compared to the other distribution. It is also important to note the very small number of preinstalled packages that come with Alpine Linux, and especially piCore.

The boot times are quite different but remain relatively fast, with the Raspberry Pi OS Lite being the longest at 1 minute 45 seconds. Looking at the number of processes running

at boot, piCore and DietPi have the fewest with 55 and 52 processes respectively, compared to 67 processes for Alpine Linux and 89 processes for Raspberry Pi OS Lite.

Following these metrics, we can see that piCore is clearly the lightest distribution, followed closely by Alpine Linux. DietPi remains lightweight, with a balance between preinstalled packages and low resource usage. Finally, Raspberry Pi OS Lite is the distribution that uses the most resources and comes with the most preinstalled packages of the four.

Distribution	Raspberry Pi OS Desktop	Raspberry Pi OS Lite	DietPi	Alpine Linux	piCore
Release date	04/07/2024	04/07/2024	07/07/2024	22/07/2024	17/04/2024
Base Linux distribution	Debian 12	Debian 12	Debian 12	Independent (originally Gentoo)	Tiny Core Linux
Community size	High	High	Average	Average	Small
Linux Kernel version	6.6.31	6.6.31	6.1.21	6.6.41	6.1.68
Image size	945.5 Mb	383.5 Mb	221.3 Mb	60.8 Mb	88.1 Mb
Disk usage	3.2 Gb	2.1 Gb	1.7 Gb	162.3 Mb	27.1 Mb
RAM usage	167 MiB	104 MiB	32 MiB	19 MiB	12.2 MiB
Preinstalled packages	1418	603	223	25	3
Boot duration	2 min 35.081s	1 min 45.225s	1 min 15.422s	32.217s	19.861s
Running processes	122	89	52	67	55

Table 4.1: Comparison between various lightweight distributions for Raspberry Pi hardware

Stat	Command
Linux Kernel version	`uname -r`
Disk usage	`findmnt -no USED / # + /boot + /boot/efi + /boot/firmware`
RAM usage	`free -tm`
Preinstalled packages	On Debian: `dpkg -get-selections | wc -l` On Alpine Linux: `apk info | wc -l` On Tiny Core Linux: `tce-status -itce-status -i`
Boot duration	`systemd-analyze time`
Running processes	`ps -e | wc -l`

Table 4.2: Commands used in table 4.1

4.3 Choice and motivations

Looking at the different metrics described in the previous section, we should choose the piCore distribution. However, as piCore is based on Tiny Core Linux, it shares the same ultra-minimalist mentality, making it difficult to access for users unfamiliar with Linux. Furthermore, its documentation is limited and its community is quite small. The decision was therefore made to discard this distribution.

The second choice was Alpine Linux. It's a more complete distribution and has a package manager similar to APT for Debian. However, although its documentation is much more comprehensive than that of piCore, and it has a relatively active community, it remains difficult to access for users unfamiliar with Linux. Moreover, after a few installation tests, it turns out that system configuration can be laborious due to compatibility issues between certain packages that are not yet fully integrated into Alpine's package manager.

The final choice was to opt for the DietPi distribution. As with the current distribution, Raspberry Pi OS Desktop, DietPi is based on Debian, thus sharing the same package manager and repositories. Additionally, it is more lightweight than Raspberry Pi OS Lite and utilises less memory, while being optimised for Raspberry hardware and relatively accessible for users with limited experience.

Chapter 5

Station configuration

This chapter will give an overview of the different steps required to configure the operating system and software of a station, as well as the different automation mechanisms that have been put in place. First, we will explain the steps required to set up the operating system and the configuration steps that can be automated with the DietPi distribution. Then the steps required to install the software prerequisites and set it up as a service so that it can be run automatically on boot will be explained, along with the setup `setup.sh` shell script that was created to automate these steps. Finally, we will present a preconfigured DietPi image that can be used to create a fully functional station, and how to create another one if needed. Table 5.1 provides an overview of the different steps automated by each method presented in this section.

	Manual installation	DietPi automated configuration	Setup script	Preconfigured image
System configuration	✗	✓	✗	✓
Additional packages	✗	✗	✓	✓
Python packages	✗	✗	✓	✓
SPI interface activation and Picamera detection	✗	✓	✓	✓
Service configuration	✗	✗	✓	✓

Table 5.1: Comparison of automation with different installation methods

5.1 System configuration

5.1.1 Manual configuration

The initial stage of the configuration and installation process is to download and install the latest version of DietPi OS on a SD card using a disk imaging tool. The steps to download and flash a SD card with the latest version of DietPi are detailed in appendix A.1.

Once the SD card is ready, before inserting it into the Raspberry Pi Zero W and proceeding with the installation, we need to enable and set up a WiFi connection:

1. Open the SD card folder `rootfs` and locate and open the following two files : `dietpi.txt` and `dietpi-wifi.txt`
2. In `dietpi.txt`, find *AUTO_SETUP_NET_WIFI_ENABLED* and change the value to 1. The line should be *AUTO_SETUP_NET_WIFI_ENABLED=1*.
3. In `dietpi-wifi.txt`, set `aWIFI_SSID[0]` to the name of an available WiFi network and `aWIFI_KEY[0]` to the password of the WiFi network.
4. Save and close both files.

Once the Wifi has been configured, the SD card can be inserted in the Raspberry Pi Zero W. For the next steps of the configuration, the user can either interact with the system by connecting a keyboard and a screen or connect via a remote SSH connection if a computer connected to the same WiFi network as registered in `dietpi-wifi.txt` is available. A guide to connect via a remote SSH connection can be found in appendix A.2.

Once the system has booted, a login prompt will appear. The default credentials are login: `root` and password: `dietpi`. DietPi will then begin to update the bundled packages. Once the update is complete, a prompt will appear asking if the user wants to enable DietPi's data collection. After either enabling or disabling data collection, the system will ask if the default password should be changed. Once the password has been changed or not, the system is ready for use. Installation of the required packages and the PhenoHive project can now continue as described in section 5.2 below.

5.1.2 DietPi automated configuration

Another advantage of using the DietPi distribution, is that it is possible to configure and automate many aspects of the system before the first boot by modifying three files: `dietpi.txt`, `dietpi-wifi.txt` and `config.txt`, which are available in the root directory of DietPi once the image has been written to the SD card.

Once DietPi has been flashed on a SD card (see appendix A.1 for more details), two partitions, which are illustrated in Figure 5.1, will be present on the disk:

- a FAT partition named `bootfs` of ~135 Mb. This partition serves as a boot partition and contains the kernel of the operating system. It will contain two folder named `dietpi` and `overlays`, as well as several binary and other text files.
- an Ext4 partition named `rootfs` of ~800 Mb. This partition will contain the system files of the operating system. It will contain folders such as `bin`, `etc`, `run`, etc.

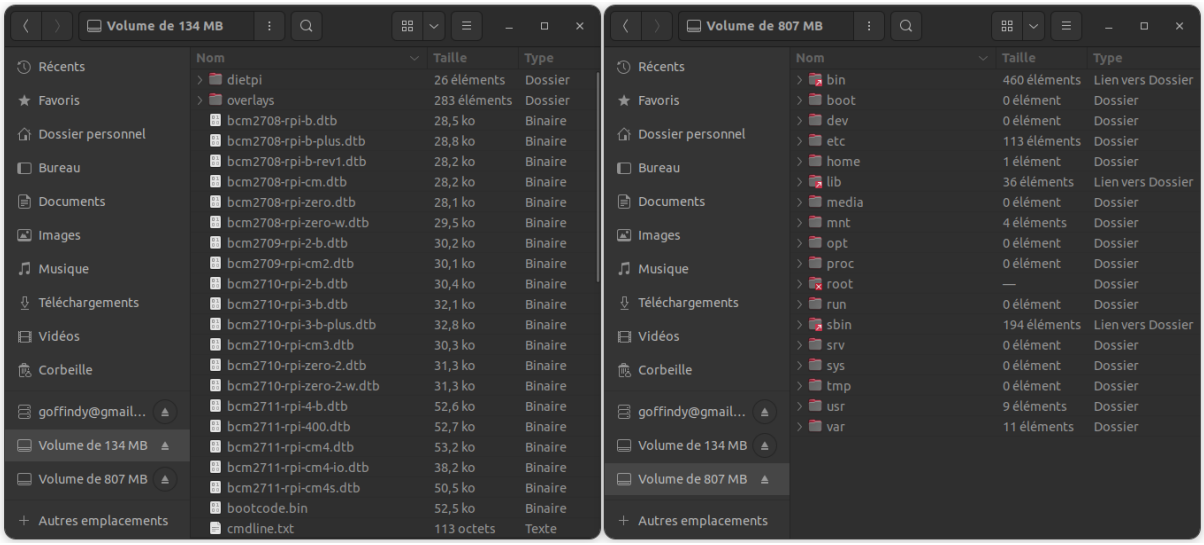

Figure 5.1: Overview of the default partitions of the DietPi distribution: `bootfs` (left) and `rootfs` (right)

The files that need to be modified are on the `bootfs` partition. They are directly available on the `tools/dietpi/` folder of the GitHub repository, with a Markdown file named `DietPi_files.md` summarising the different modifications. A rendered version of this Markdown file is provided in appendix G. This section will explain the different changes made to each file.

`dietpi.txt`

This file contains many variables that are employed to configure the system upon initial boot. It is divided into several sections; the present discussion will focus on the sections and variables that have been modified to align with the specific requirements of a PhenoHive station.

- `Language/Regional options`: the values changed in this section define the default keyboard layout and timezone. The keyboard was changed from the English QWERTY layout to the French AZERTY one, and the timezone was changed from Europe/London to Europe/Brussels.

- `Network options`: the values of this section were changed to enable WiFi and disable Ethernet connection on boot. The system will then look for a WiFi connection on boot, as explained later. The hostname of the system was also changed from DietPi to PhenoHive, the country code was changed from Great Britain to Belgium, and the delay at boot until network connection is established was disabled as otherwise the system would wait at each boot for a network connection even if none are available.
- `Software options`: here, the only value changed was to switch the default SSH server from Dropbear to OpenSSH, which offer more comprehensive features and better security updates at the cost of being a bit more heavy.
- `Non-interactive first run setup`: this section enables the system to update, run the initial setup and install additional software packages on first boot without user input. Furthermore, it is here that the default password can be changed; for this version of the system, it was changed from `dietpi` to `phenohive`. In this section, it is also possible to specify additional software to install through `dietpi-software`, to speed up the software install process, the following packages were added: `git`, `RPi.GPIO`, `InfluxDB`, `Grafana`, `Python 3` and `Python 3 pip`.
- `Misc DietPi program settings` and `DietPi-Config settings` sections: in these last two section, the variable were set to opt-out of DietPi survey, and to disable the automatic daily check for system and packages updates to prevent the system from changing without user input.

`dietpi-wifi.txt`

This file allows the configuration of WiFi connections that the system can connect to on boot. It requires WiFi to be enabled on the system via `dietpi.txt`. It can store up to five entries, where one entry correspond to one WiFi network. For each entry, two fields need to be changed. As an example, a default WiFi connection named `PhenoHive` was enabled, by modifying the following fields:

- SSID (name): `aWIFI_SSID[0]='PhenoHive'`
- Key (password): `aWIFI_SSID[0]='PhenoHive'`

`config.txt`

This file contains variables specific to the Raspberry Pi hardware. It allows enabling of certain interfaces and connections by default. Two modifications were made:

- The Picamera module and detection was enabled by adding `start_x=1`, which start the Picamera module, and `camera_auto_detect=1` to allow the system to automatically detect the correct Picamera module version and use the corresponding drivers.

- The Serial Peripheral Interface (SPI) interface was enabled using `dtparam-spi=on`. The SPI is a communication protocol used to communicate with the display and that needs to be enabled on the Raspberry Pi.

These and other parameters can also be changed in the `dietpi-config` software once the first boot has been completed. However, this allows to enable them without any user intervention.

5.2 Software configuration

The first step to facilitate the installation of the stations was to provide an automated method to configure the system, install the required packages, and set up the PhenoHive software. To fulfil this goal, a `setup.sh` shell script was created.

5.2.1 Manual configuration

Cloning the repository

The initial step in the installation process is the installation of the `Git` software. `Git` is an open-source tool that enables version control in the development process of a project. Furthermore, it allows developers to collaborate on the same project from different computers without causing conflicts. In this case, it allows the project to be downloaded from a public repository hosted on GitHub. `Git` can be installed using the `APT` packet manager of DietPi and other Debian-based distributions with the following command:

```
sudo apt-get install git
```

Once `Git` has been installed, the repository can be downloaded in the `/root/PhenoHive` folder using the following command:

```
git clone https://github.com/Oldgram/PhenoHive.git /root/PhenoHive
```

Installing required packages

Before proceeding to the installation of the Python libraries needed to run the project, it is crucial to first install several packages using the following command:

```
sudo apt-get update
sudo apt-get install build-essential cmake gfortran pkg-config git python3
↪ python3-pip python-is-python3 libopenblas-dev libatlas-base-dev patchelf
↪ ninja-build libcap-dev ffmpeg python3-dev python3-smbus python3-pil
↪ python3-rpi.gpio python3-av python3-libcamera python3-kms++ python3-pyqt5
↪ python3-prctl python3-numpy python3-scipy python3-sklearn python3-skimage
↪ python3-statsmodels
```

In the different packages installed above, some are Python packages. Although it is generally recommended to install Python packages using the Python package manager `pip` to avoid conflicts between packages installed by the system package manager and `pip`, no other project is installed on a station. Therefore, conflicts are easily avoided.

Moreover, the installation of Python packages via the system packages is significantly faster than that via pip, as the former are optimised for the distribution, whereas the latter will require the building of each package prior to installation. This process can be quite lengthy due to the limited hardware capacity of the Raspberry Pi Zero W.

Nevertheless, certain packages must be installed via pip, either because they are not available in the Debian repositories (as it is the case with the `picamera2` package) or because we need a specific version is only available through pip (as it is the case with the `OpenCV` package).

Installing required Python libraries

It is crucial to install a specific version of each package, as PlantCV, the library used for processing plant images, is only compatible with particular versions of certain libraries. Furthermore, before proceeding with the installation, it is essential to assert the compatibility of the required versions with both the Raspberry Pi hardware and the current Python version (on Debian 12 Bookworm, the Python version in use is 3.11). This can be verified using the piwheels.org website.

It is also important to consider that the Raspberry Pi Zero W is based on a 32-bit architecture, which can result in compatibility issues. This is because some software may require more memory address space than is available from a 32-bit architecture. This is particularly evident in the case of PlantCV, whose version past v4.0, released on the 21st of August 2023, has become incompatible with 32-bit systems. Consequently, the latest version compatible with 32-bit architecture is v3.14.3, released on the 30th of March 2023.

By taking this into accounts, the different packages and their respective version that are compatible as of the moment writing this report are can be installed with the following command:

```
sudo pip install configparser==7.0.0 influxdb-client==1.44.0 hx711==1.1.2.3
↪ pandas==1.5.1 adafruit-gpio==1.0.3 picamera2==0.3.18 opencv-python==4.7.0.72
↪ plantcv==3.14.3 mizani==0.9.0 plotnine==0.10.1 --break-system-package
```

It is important to note that the flag `-break-system-package` must be used when running `pip` with administrator privileges. This is due to the potential for conflicts between packages installed with the system package manager and those installed with `pip`. However, as previously mentioned, this is not a significant issue as no other Python packages will be installed on the system.

The last library to be installed is used to communicate with the ST7735 display. It is not possible to install this library using `pip`; instead, it must be downloaded from the GitHub repository. The command to download and install the libraries is as follows:

```
git clone https://github.com/degzero/Python_ST7735.git
cd Python_ST7735
python3 setup.py install
```

Enabling SPI interface and Picamera detection

Finally, it is necessary to enable the SPI interface and the Picamera detection. This can be done using the DietPi configuration interface. It can be accessed using the following command:

```
sudo dietpi-config
```

Once the interface is open, access the `Display Options` menu, and enable the RPi Camera.

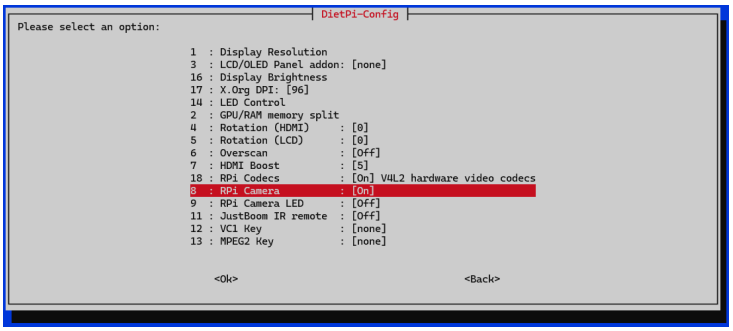

Figure 5.2: Dietpi configuration menu — Picamera activation

Next, navigate back to the main menu, access the `Advanced Options` menu, and enable the SPI interface.

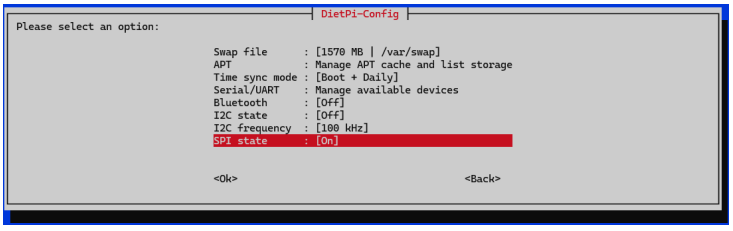

Figure 5.3: Dietpi configuration menu — SPI activation

Service configuration

The final phase of the installation process is to configure the software as a service. This enables the operating system to execute the software at system startup. On Linux, a service, also known as a `daemon`, is a background process that is run by the operating system and that operates autonomously. These processes start at system boot and continue running until the system is shut down or until a specific policy, as defined in their corresponding `service` file [36].

As with other distributions based on Debian, DietPi employs `systemd` as its system and service manager. In order to configure PhenoHive as a service utilising `systemd`, it is necessary to create a file named `phenohive.service` within the directory `/lib/systemd/system`. This can be done by executing the following command:

```
sudo nano /lib/systemd/system/phenohive.service
```

The command shown above uses `nano`, a simple command-line text editor, to create and modify the `phenohive.service` file. Upon opening the file in the nano text editor, a blank document will be displayed which need to be filed with the following content:

```
1  [Unit]
2  Description=PhenoHive station service
3  Documentation=https://github.com/Oldgram/PhenoHive
4  After=multi-user.target
5
6  [Service]
7  User=root
8  WorkingDirectory=/root/PhenoHive
9  ExecStart=/usr/bin/python /root/PhenoHive/main.py
10 Restart=on-failure
11
12 [Install]
13 WantedBy=multi-user.target
14 Alias=phenoHive.service
```

Figure 5.4: Content of the `phenohive.service` file

This file is divided into three parts:

- The `Unit` section provides a brief description of the service, a link to the service's documentation (in this case, the GitHub repository of the project), and indicates when the service can be launched (in this case, after the operating system has been fully initialised).

- The `Service` part indicates that the service should run as the user `root`. It sets its working directory as the root of the project, and specifies the command to start the service. Finally, it sets the restart policy for the service, which here is `on-failure`, meaning that the system will try to restart the service if it crashes.
- The `Install` part provides an alternative name for the service, and indicates that the service has been set up as a `multi-user` service, which mean that the service can be run once the system is running and the network services are enabled.

Once the service file has been configured, it must then be activated using the `systemd` service manager. This may be achieved by executing the following commands:

```
sudo chmod 644 /lib/systemd/system/phenohive.service
sudo chmod +x /root/PhenoHive/main.py
sudo systemctl daemon-reload
sudo systemctl enable phenohive.service
```

The first command modifies the file permissions for the `phenohive.service` file, assigning read and write permissions to the current user. The second command grants executable permissions to the Python script, located at `/root/PhenoHive/main.py`, which is responsible for executing the PhenoHive software. The third command reloads the `systemd` manager configuration, causing `systemd` to re-read all unit files and ensuring that any changes made to the service files are recognised. Finally, the last command enables the `phenohive.service` to start automatically at boot time.

Database configuration

Another step in configuring the system is to edit the `config.ini` to update the station ID and the different information used to connect to InfluxDB. The fields that need to be updated are the following:

```
[Station]
# Unique ID of the station
id = 1

[InfluxDB]
# InfluxDB token
token = InfluxDB Token
# InfluxDB organization
org = PhenoHive
# InfluxDB bucket to store the data
bucket = PhenoHive_data
# Url of the server running InfluxDB
url = http://10.42.0.1:8086
```

- The ID of the station serves to identify to which station corresponds the data send to the InfluxDB database. It must be unique for each station and can be any value, such as 1, group_a, etc.
- The InfluxDB token is an access key that authorises stations to send data, given at the creation of the database on the InfluxDB server.
- The organisation corresponds to a workspace in InfluxDB created for the occasion, which includes the various data tables.
- The bucket, created during the database configuration, corresponds to the table in which the data is stored.
- The URL is the address at which the InfluxDB server can be found on the network.

Load cell calibration

The last step in the configuration is to run to compute the calibration coefficient. It can be done in two ways:

- By running the station's software and using the graphical interface to navigate to the calibration menu. Once inside the calibration menu, place a weight that is identical to the value stored in the configuration file (by default, a weight of 1.5 kg) and calculate the coefficient by pressing the *Get Calib* (left) button.
- By running the `tools/calibration.py` Python script. This script will prompt the user to place a known weight on the load cell, and to enter its weight in grams to calculate its calibration coefficient. This coefficient allows the raw data measured by the load cell to be converted into grams.

The coefficient is calculated using the following equation:

$$\text{coefficient} = \frac{\text{real weight in grams}}{\text{measured weight}}$$

Once the coefficient has been found, it is stored in the `config.ini` file at the root of the software directory. This coefficient does not change even if the station is powered down. However, it is unique to each load cell, thus the calibration must be done on each station.

5.2.2 `setup.sh` script

In order to automate the setup of a new station, the different steps described above have been incorporated into a shell script named `setup.sh`. A shell script is a text file containing a sequence of commands for a UNIX-based operating system.

This script is available at the root directory of the repository. Once the repository has been downloaded using the `git clone` command as described above, it can be run with the following command:

```
sudo bash setup.sh
```

In order to facilitate the monitoring of the script's progression, messages have been integrated into the script to provide the user with updates on the status of completed and ongoing steps. Figure 5.5 illustrates a typical output of the script.

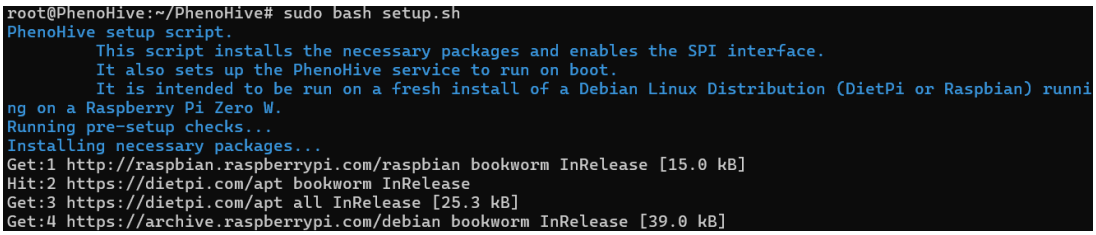

Figure 5.5: Example of an output produced by the `setup.sh` script

The `setup.sh` script follows these steps:

1. It runs a few checks before it starts, checking that the system is connected to the Internet, that it is running with administrator privileges, and that it is running from the root of the PhenoHive directory.

2. It installs the required system packages by loading the `tools/setup.config` files, which contain the list of required packages. This enables a certain degree of modularity, as the required packages can be changed without having to modify the script in itself.

3. Similarly, it installs the required Python packages as specified in the `requirements.txt` file.

4. It automatically downloads and install the ST7735 library.

5. Once the required packages have been installed, it enables the SPI interface and the Picamera recognition by modifying the `config.txt` file.

6. Finally, it setups the system by copying the file `phenohive.service` file located in the `tools/` folder to the `/lib/systemd/system/` directory. It then reloads the daemons and enables the service.

7. Once the setup is complete, it will display the following message to: "Setup complete. A reboot is required before running the service.".

The only steps that were not automated were the modification of the configuration file `config.ini` to align with the current configuration, and the calculation of the calibration coefficient of the load cell. As described above, the calibration coefficient can be calculated using either the provided `tools/calibration.py` Python script or directly via the user interface of the station, and the configuration file `config.ini` must be modified manually to complete the installation process.

5.3 Preconfigured system

5.3.1 Usage

The last automation method that was put in place to facilitate the setup of the station was to provide a fully preconfigured image of the system. The image weights 8 GB and can be downloaded from here[1].

This image can be written to SD cards of at least 8 GB and can be used on larger SD cards as the DietPi system will automatically expand the file system to use all available space.

The steps used to create this image and how to create a new image from a preconfigured system are explained in section 5.3.2 below.

By using this image to flash a new SD card, it only needs to be inserted in the Raspberry Pi Zero W to be fully operational.

The image will automatically connect to a WiFi network named `PhenoHive` and a password `PhenoHive`, as in the examples described above. The repository will already be available

The only steps remaining are to ensure that the project is up-to-date using the following command inside the repository of the project (located in the `/root/PhenoHive/` folder):

```
sudo git pull
```

Once the project is up-to-date, the configuration file `config.ini` must be updated to fit the new configuration. The calibration coefficient must also be calculated using either the provided calibration script or directly via the user interface of the station.

5.3.2 Image creation

The image was created by cloning an 8 GB SD card containing a fully operational PhenoHive system running on DietPI 9.6. It was created using the `disk dump` (dd) software. Disk

[1] `https://uclouvain-my.sharepoint.com/:u:/g/personal/dylan_goffinet_student_uclouvain_be/Eb_mK8L4GptGhppHbfrdEPoBmDWEppfjG-rZpERCGurvsw?e=Wdi3cV`

dump is a command-line utility available in Unix operating system used for converting and copying files.

The command used to create the image was the following:

```
sudo dd bs=4M if=/dev/sdb of=path/to/DietPi_9.1.img status=progress
```

- `bs` is the block size in megabytes. This determines how much data `dd` reads and writes at a time.
- `if=/dev/sdb/` correspond to the input file (`if`), which is the path to the sdb card to be copied.
- `of=path/to/DietPi_9.1.img` correspond to the output file (`of`), which is the path where the image will be saved.
- `status=progress` tells dd to displays progress information during the operation.

This command creates a complete bit-by-bit copy of the file system, including the 'empty space'. However, if the output image is compressed into an archive (.zip for example), the white space is automatically shrunk, resulting in a lighter archive. For example, the 8 GB copy provided earlier was reduced to 2.56 GB when compressed into an archive.

It is important to note that an image copied from a 16 GB SD card can only be written to a 16 GB or larger SD card, as 16 GB of data will be written to the destination. However, as explained earlier, it can be used on larger SD cards as the DietPi system will automatically expand the file system to use all available space on first boot.

Chapter 6

Implementation

This chapter will examine the modifications and enhancements applied to the software with the objective of enhancing its readability, maintainability, modularity and overall robustness.

6.1 Project layout

The changes mean that the project layout is different from the original layout received in September. The following sections explain in more detail why the changes have been made and what they mean for the software. Before doing so, however, it is important to understand the new project layout.

Figure 6.1 shows an image of the list of files contained in the GitHub repository[1].

[1]The GitHub repository is available at the following address: `https://github.com/Oldgram/PhenoHive`

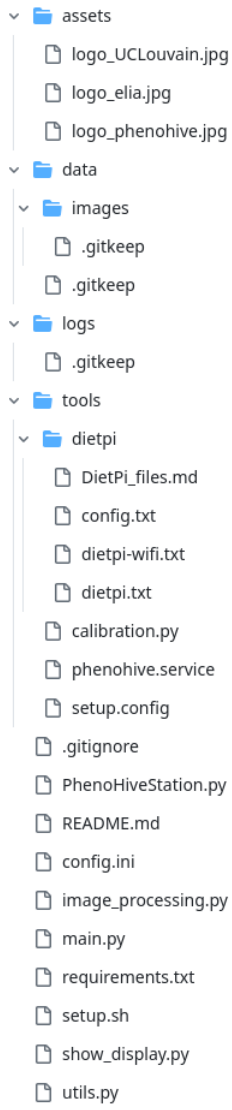

Figure 6.1: Content of the GitHub repository

The repository contains several folders. The `assets/` folder contains the various images displayed on the screen and in the `README.md` file.

`data/` is used to store the Comma Separated Values (CSV) file containing the different measurements (such as the plant weight) taken by the station. `data/images/` is used to store the various pictures taken by the station. The `logs/` folder is used to the different logs generated by the software logging system.

The `tools/` folder contains utility files such as the `calibration.py` Python script, which is used to determine the calibration coefficient of the load cell to convert the raw data to real weight (in grams). It also contains files used during the installation process, such

as the `phenohive.service` file, which allows the operating system to automatically start the software on startup, and the `setup.config` file, which contains the Linux packages required to run the software. `tools/dietpi/` contains several DietPi OS configuration files, which were discussed earlier.

The root of the repository contains five Python scripts, which collectively constitute the software that runs the station. The respective roles and contents of these scripts are as follows:

- `PhenoHiveStation.py` contains the *PhenoHiveStation* class, which is responsible for handling the different measurement pipelines of the software.
- `image_processing.py` contains the different functions responsible for processing the images taken by the station.
- `main.py` is responsible for initialising the station and managing the different phases of the software.
- `show_display.py` contains the *Display* class used to communicate and display the different menus on the screen.
- `utils.py` contains the functions used to initialise the logging system and to write the various measurements to CSV files.

Alongside these Python scripts are also located three important files:

- `config.ini` contains the different parameters of the station, such as the station ID, the different General-Purpose Input/Output (GPIO) pins to use for which sensors, the credentials to communicate with InfluxDB, and others. These parameters will be discussed in more detail later.
- `requirements.txt` specifies the different Python packages and versions required to run the system.
- `setup.sh` is a shell script[2] used to automatically install the various programs and packages required to run the software. Its operation was explained in the previous chapter.

There is also a Markdown[3] file in the repository root. The `README.md` file contains a documentation of the project and its installation process. It is automatically rendered

[2]A shell script is a text file containing a sequence of commands for a UNIX-based operating system. It enables users to automate repetitive tasks by executing the script instead of typing each command manually.

[3]Markdown is a lightweight markup language created by John Gruber in collaboration with Aaron Swart in 2004. It features plain text formatting syntax and is designed to be converted into formats such as HTML.

when the repository page is opened on GitHub, and is used to give an overview of the project without having to read its implementation.

The repository also contains several git-specific files, such as `.gitignore` which allows you to specify files and directories that Git should ignore [37]. There are also several `.gitkeep`, which are empty files used to add empty directories to the repository. While this it is not an official solution, it is a commonly accepted way of keeping track of empty directories that are normally ignored [38]. In our case, they serve to indicate the expected architecture of the project, but are not necessarily needed, as their path can be changed in the configuration file, and they are automatically created by the software at boot time if they don't exist, to avoid errors when the software tries to save a file in a folder that doesn't exist.

6.2 Documentation

6.2.1 Internal documentation

In order to achieve a code that is both readable and maintainable, it is essential to ensure that the documentation is consistent throughout the various sections of the code. Furthermore, this approach serves two main objectives:

- Ensure that the user uses the application as it was designed.
- Ensures that knowledge is passed correctly between developers. This enables anyone wishing to modify or add functionalities to the application to do so as easily as possible.

As it was received in September, the documentation in the code was limited to a few comments at specific stages of the program's life. Additionally, the code was only separated into three files:

- `main.py`, which represented most of the program. It was responsible for loading the configuration file, initialising the station's various components, navigating the different menus, and taking data (weight collection, taking the picture, sending the data to the database) when the station entered measurement mode.
- `image_processing.py` contained the different functions to process the collected photos and compute the growth of the maize plant by establishing its skeleton using `PlantCV`, and then computing the sum of its segments.
- `show_display.py` contained the *Display* class, containing the different methods to display the information on the LCD screen.

As we will see in the section 6.3, the code has been broken down into smaller functions and methods. Each of these functions and methods has been documented using docstrings.

Docstrings are strings of documentation within the code that make it easy to understand a piece of code (usually functions, methods and classes). They allow the developer to understand what a function does without having to read the details of the implementation [39]. They are generally used for code sharing and to allow developers to remember their code when they need to return to it.

A variety of docstrings formats are commonly employed within in the Python community, including Epytext style, Google style, and NumPy style which is commonly used in scientific computing and data analysis projects [40]. It was determined that the Sphinx/reStructuredText (reST) style would be the optimal choice, as it is a prevalent preference among the Python community and is natively supported by numerous Integrated Development Environments (IDEs), including the JetBrains IDEs and Visual Studio Code. Moreover, it can be generated using Sphinx [41], and is the format recommended by the Python Enhancement Proposals (PEP) 287 [42].

Furthermore, type hinting [43] has been incorporated into function and method declarations to indicate the anticipated types of the various parameters, as well as the expected return. Type hints were introduced in Python version 3.5 and serve to provide hints to developers and tools with information regarding the types of arguments a function or method expects and the type it returns.

It should be noted that Python does not enforce type hints at runtime; they are solely intended for documentation and tooling purposes. Moreover, they are supported by a range of tools, including IDEs, linters[4] and type checkers, which facilitate enhanced code analysis and error checking.

Figure 6.2 shows an example of a function that uses type hints and Sphinx/reST-style docstring:

```
1 def get_segment_list(image_path: str, channel: str = 'k', kernel_size: int = 20) ->
  ↪ list[int]:
2     """
3     Get the list of segments lengths from the plant skeleton
4     :param image_path: path to the image
5     :param channel: CMYK channel for conversion from RGB to CMYK colorspace (c = cyan,
      ↪  m = magenta, y = yellow, k=black) (default = 'k')
6     :param kernel_size: kernel size for the closing operation (default = 20)
7     :raises: KeyError if no segments are found in the image
8     :return: list of segments lengths
9     """
```

Figure 6.2: Python function featuring type hints and reST-style docstring

[4] A linter is a tool that analyses source code to identify and flag programming errors, bugs, stylistic issues, and other suspicious constructs to help improve overall code quality

6.2.2 External documentation

In addition to the internal code documentation, an external documentation in the form of a Markdown `README.md` file was created to provide an overview of the system, its usage and how it works without the need to read the implementation. The file is divided into three sections: Project Description, System Operation and Installation.

The first section, *Project Description*, provides a brief introduction to the PhenoHive project. It describes the various hardware components used and gives a concise explanation of the function of each Python script utilised by the station.

The second section, *System Operation*, provides an overview of the operation of the system, with a breakdown of its life cycle. It begins with an explanation of the configuration and initialisation process and continues with a description of the station's various menus and measurement mode. It also describes the logging and error handling mechanism, which will be discussed in section 6.4.

The third and final section, *Installation*, focuses on the various steps required to set up a station, from the operating system to the project setup. It provides comprehensive instructions for installing the operating system via a manual installation, a pre-built image or by using a semi-automatic installation of the OS through configuration files modified for the project. The modifications made to the configuration files are described in the `DietPi_files.md` file. This section also describes how to set up the project, either by using the automated setup script created during this work, or manually.

A rendered version of the Markdown file `README.md` is available in Appendix F.

6.3 Refactoring

As mentioned earlier, the code as it was received in September was only separated into three Python scripts, with `main.py` being the file that managed the majority of the station's program life cycle. Additionally, the code was divided into a limited number of functions and contained a number of hard-coded values, which made it challenging to implement any modifications or additions to the code.

In order to enhance the maintainability, readability, and modularity of the code, a refactoring phase was undertaken. Refactoring is defined as the process of modifying the internal structure of the software with the objective of facilitating comprehensibility and facilitating modification at a lower cost, while maintaining its external behaviour.

The first step was to divide the existing functions into smaller, more manageable units. One example of such changes is the `main` function, which previously consisted of a loop that enabled navigation between the different menus using nested loops and conditions. This function was kept, but divided into smaller, more dedicated functions

such as the `handle_configuration_menu` function that enables navigation within the calibration menu. As illustrated in Figure 6.3, two methods were extracted from the `measurement_pipeline` method, `picture_pipeline` and `weight_pipeline`. This improves overall code readability and maintainability by keeping smaller, specialised functions for specific aspects of the code. It also improves the modularity of the measurement pipeline, as the pipeline can be extended if necessary to include the data collection of an additional sensor without the need for complex modifications.

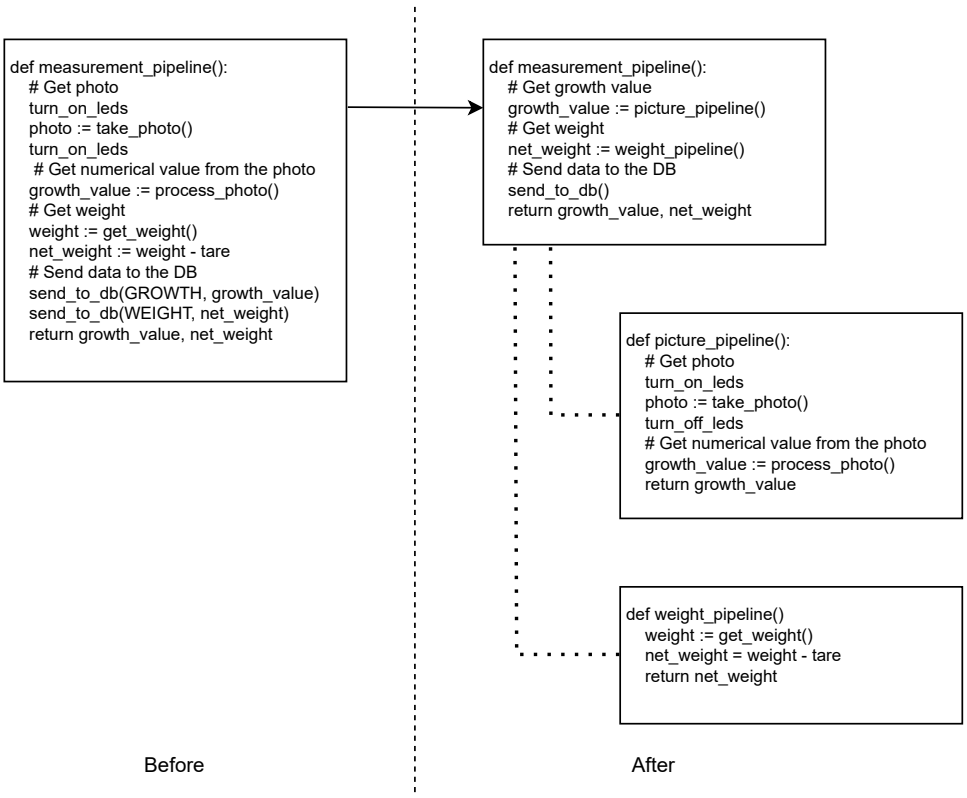

Figure 6.3: Extract refactoring method used within the measurement pipeline

The second step was to make use of Object-Oriented Programming (OOP) in order to encapsulate data and functions, thereby avoiding the use of hard-coded or repeating variables. This encapsulation renders the necessity for global variables obsolete and guarantees the consistency of the data throughout the code.

To this end, a `PhenoHiveStation` and a `Display` class have been created to centralise the different variables and methods that the station may require. Additionally, for consistency and ease of access, and given that each station only requires a single instance of the `PhenoHiveStation` class to be running at any given time, it was implemented as a singleton.

A singleton is a creative design pattern that ensures that only one instance of a class can be running at any one time. Furthermore, lazy instantiation was employed to guarantee that the object is only created when it is required [44]. In `PhenoHiveStation`, lazy instantiation was implemented by deferring the creation of its instance until the `get_instance()` method is called for the first time.

As previously stated, a singleton serves several purposes, including centralisation. Moreover, it ensures that critical operations are only run once at a time. For example, it allows for the maintenance of only one database connection throughout the execution of the software.

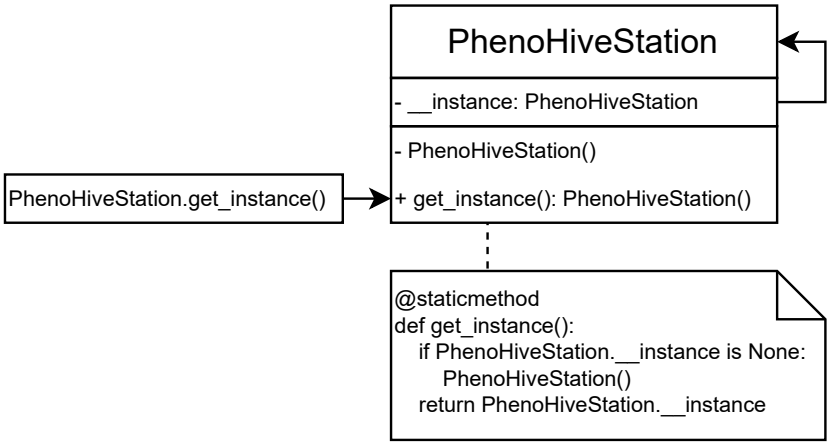

Figure 6.4: Structure of the PhenoHiveStation class as a Singleton pattern

The final step was to further utilise the configuration file `config.ini` in order to remove the last remaining hard-coded values. The configuration file had already been employed for the majority of the variables, with values pasted using the `configparser` Python library [45].

However, some values remained directly embedded within the source code of the program. This resulted in inflexibility and maintenance challenges. If a value needed to be changed, all instances of that value would require updating, which could introduce bugs if some instances were missed. The incorporation of these values into the configuration file would therefore improve the overall modularity of the software and facilitate adaptation to other environments.

Below are the parameters added within `config.ini`.

```
[Paths]
data_folder = data/
image_folder = data/images/
csv_path = data/measurements.csv
log_folder = logs/
```

```
6
7  [Display]
8  width = 128
9  height = 160
10 speed_hz = 4000000
11 dc = 24
12 rst = 25
13 spi_port = 0
14 spi_device = 0
15
16 [Camera]
17 led = 23
18
19 [Buttons]
20 left = 21
21 right = 16
```

The `Paths` section contains relative paths to the `images` folder, where the images captured by the station are stored, to the CSV file that will be used to save the different measurements, and to the `logs` folder where the logs files generated by the station will be stored. It is important to maintain relative paths, as this allows a greater flexibility and portability compared to absolute paths, which may not be accurate depending on the system architecture.

For instance, if the project is moved to a different directory or another machine, relative paths will still function as long as the project structure remains consistent. In contrast, absolute paths will need to be modified to align with the new environment.

The `Display` section contains the different attributes of the ST7735 display, namely its width and height, the speed of the Serial Peripheral Interface (SPI) communication and the different GPIO pins used to communicate with the display. Similarly, the `Camera` and `Buttons` sections contain the different GPIO pins used to communicate with the LED lighting strip and the left and right buttons.

6.4 Logging and error handling

The implementation of an appropriate logging system can be highly beneficial when developing an application, particularly when it is designed to operate on a dedicated machine that may be challenging to monitor in real-time. Furthermore, logging provides a mechanism to track events and operations within the software, enabling a more comprehensive understanding of its behaviour. It also offers valuable insight if the system encounters unexpected behaviour, making it easier to diagnose what went wrong and resolve the problem.

To this end, a logging mechanism was implemented using the `logging` Python library, which has been part of the standard Python Library since version 2.3. One key benefit of this module is that it that once a logger has been created, it can be used by the different parts of the software by calling `logging.getLogger(__name__)`. It also allows setting up formatting options to the outputs, such as the time and date, to provide a more comprehensive timeline of the events, as well as different logging levels.

Figure 6.5 shows the output produced by the logging system from the initialisation of a station to the end of a measurement cycle. As can be seen, several logging levels are used, such as `INFO`, `DEBUG` and `ERROR`, which serve different purposes. As can be observed on line 10, the application has encountered an error when processing the photo taken during the measurement cycle and provides information on the possible source of the problem (in this case, the station was not correctly positioned in front of the plant). As mentioned in section 6.1, these logs are stored in the `logs/` folder and can be retrieved using a remote SSH connection.

```
1  2024-07-03 15:19:34,089 - PhenoHiveStation - INFO - Initializing the station
2  2024-07-03 15:19:34,200 - PhenoHiveStation - DEBUG - InfluxDB client initialized with
   ↪ url : http://10.42.0.1:8086, org : PhenoHive and token :
   ↪ xp93r5qI8i7Fdl-4f00j33257RV2CTNpj_bPyMXKE-0GLaW_kjZQ1aLuRZa5wvOm8U2TIBv_nkidbv8A==,
   ↪ Ping returned : \cmark
3  2024-07-03 15:19:34,201 - PhenoHiveStation - DEBUG - Initializing screen
4  2024-07-03 15:19:38,099 - PhenoHiveStation - DEBUG - Resetting HX711
5  2024-07-03 15:19:39,133 - PhenoHiveStation - DEBUG - HX711 reset
6  2024-07-03 15:20:41,161 - PhenoHiveStation - INFO - Entering measurement loop
7  2024-07-03 15:21:44,975 - PhenoHiveStation - INFO - Measuring time reached, starting
   ↪ measurement
8  2024-07-03 15:21:45,144 - PhenoHiveStation - INFO - Starting measurement pipeline
9  2024-07-03 15:21:54,726 - PhenoHiveStation - DEBUG - Photo taken and saved at
   ↪ /root/PhenoHive/data/images/2024-07-03_15-21-52.jpg
10 2024-07-03 15:22:03,429 - PhenoHiveStation - ERROR - KeyError: Error while processing
   ↪ the photo, no segment found in the image. Check that the plant is clearly visible.
11 2024-07-03 15:33:38,058 - PhenoHiveStation - DEBUG - Weight (median) : 2011023.4 in
   ↪ 689.2919020652771s
12 2024-07-03 15:33:40,452 - PhenoHiveStation - DEBUG - Sending data to the DB with field
   ↪ ID : StationID_2
13 2024-07-03 15:33:40,497 - PhenoHiveStation - DEBUG - Data sent to the DB
14 2024-07-03 15:33:42,638 - PhenoHiveStation - INFO - Measurement pipeline finished
```

Figure 6.5: Example of logs produced by the software

Another important mechanism to put in place alongside a logging system is proper error handling. This serves several purposes:

- Prevent the software from stopping unexpectedly if something goes wrong.
- Provide a better understanding of the potential problems that the application may face.
- Catch any errors that may occur in the code, from simple typos to unexpected behaviour.

In Python, error handling is done by encapsulating pieces of code inside a `try/except` statement, where the software tries to execute the code contained in the `try` statement, and if an error or exception occurs, it is caught by the `except` statement. It is also possible to specify the type of error it is expected to catch, providing a more precise indication of the potential source of the problem.

```
1 try:
2     growth_value = get_total_length(image_path=path_img, channel=self.channel,
      ↪  kernel_size=self.kernel_size)
3 except KeyError:
4     self.register_error(KeyError("Error while processing the photo, no segment found
      ↪  in the image. Check that the plant is clearly visible."))
```

Figure 6.6: Example of a `try/except` statement

When integrated with an effective logging system, error handling enables the generation of precise error messages and the identification of the probable cause of the problem. As illustrated in Figure 6.6, the function `get_total_length` may raise a `KeyError` if no segments are found within the image.

It was important to build error handling into the software. As it is intended to run for hours or days without human intervention, it was crucial to have the ability to trace and diagnose any errors that might occur.

Additionally, the operating system is configured to restart the software in the event of a failure or crash, which could be caused by an unexpected error. This made debugging a challenging process, as there were no logs of the program's output to assist in tracing the previous error and its underlying cause.

In order to properly register the various errors encountered during the execution of the software, a `register_error` method has been created within the `PhenoHiveStation` class. The method accepts the exception in question as an argument, logs it using the system logger, updates the station's status to indicate that an error has occurred, and stores its details within the measurements that are saved in the CSV file and sent to the database. Details of the different data saved and sent to the database are explained in section 6.6.

6.5 Interface

Some changes were made to the different menus available through the user interface of the station. Figure 6.7 show the different menus available and the navigation paths between each screen.

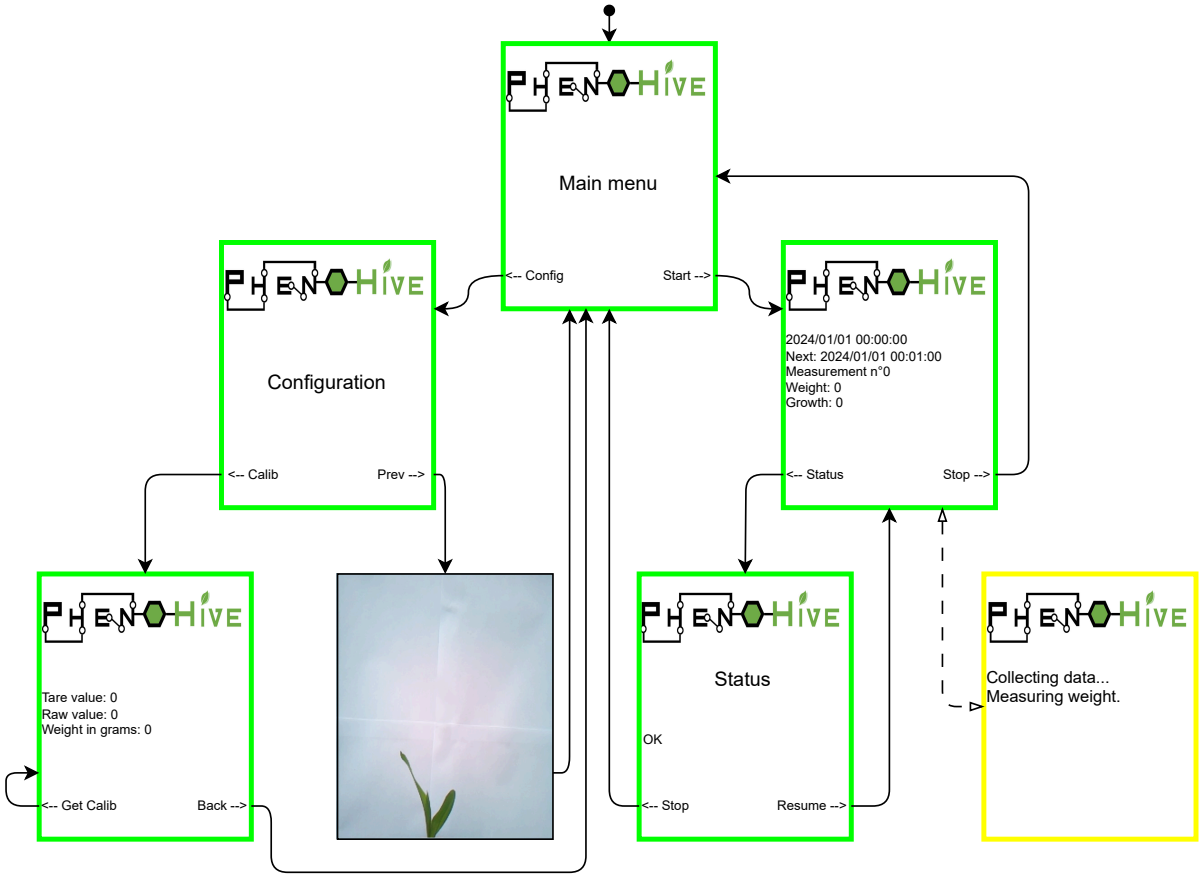

Figure 6.7: Navigation paths between the different menus

A key change is the new coloured outline around each screen to indicate the current status of the station. The colours available are:

- Green: indicates that the station is functioning as intended.
- Blue: indicates that the station is not connected to the database, but is otherwise functioning as intended.
- Yellow: indicates that the station is busy collecting and processing data. This will be the case during the measurement pipeline.
- Red: indicates that the station encountered an error.

The station selects the correct colour using the variable `PhenoHiveStation.status`, which can have three possible values: -1 on error, 0 if in the measurement pipeline and 1 otherwise. If the value is 1, the station will check if it is currently connected to the database to choose between green or blue.

Additionally, more information have been added to several screens. Namely:

- The *calibration* menu now displays the net value, i.e. the value measured by the load cell minus the tare. In addition, the tare is now calculated when the calibration menu is entered, stored in the configuration file and used until the calibration menu is entered again.
- The *measurement* menu now shows the number of measurement cycle done since the last boot.
- The *measuring* screen now provides more information concerning the current phase of the measuring pipeline. For example, it now indicated "Measuring weight", "Taking photo", etc. instead of simply indicating "Collecting data". This has been added as an indication of whether the station has been stopped at a particular stage or is still in progress through the measurement pipeline.

Furthermore, a *status* menu has been incorporated to provide additional information concerning the status of the station. This menu, which can be accessed from the *measurement* menu without disrupting the measurement cycle, displays the current status of the station and, in the event of an error, the error encountered and the time at which it occurred. Once in the menu, the user can choose to either continue the measurement cycle or stop the current measurement cycle and return to the main menu. Stopping the measurement cycle can be useful in the event of a recurring error. For example, if the station cannot communicate properly with the load cell, it may be necessary to check the station's circuitry, or if the station cannot process the image correctly because the installation is no longer fully visible to the camera and the station needs to be repositioned.

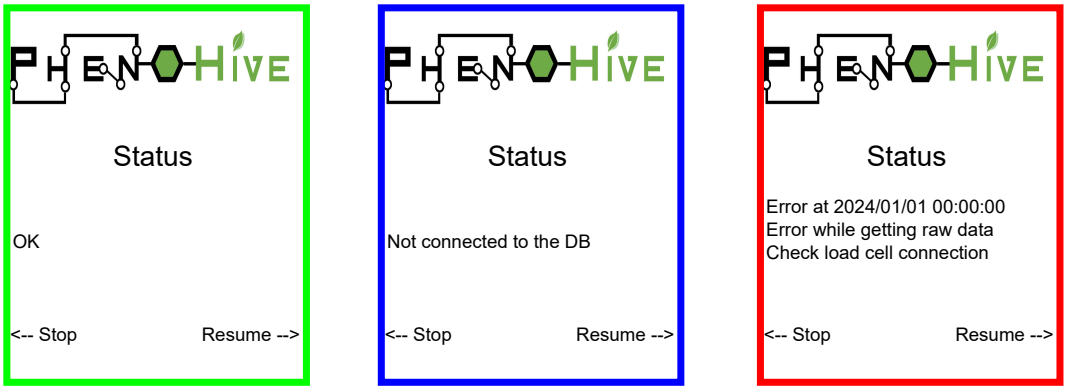

Figure 6.8: Example of different status screens

6.6 Data collection

The stations were initially designed to operate continuously as a network, with an access point connected at all times. This access point serves as a server, to which the data collected is transmitted. However, this design did not account for potential communication errors between the stations and the server, nor for the possibility of the server becoming unavailable, which would render the system inoperable.

To address this issue and ensure a certain level of data redundancy, the functioning of the stations was slightly modified to enable them to operate in an offline mode. At the end of each cycle, the station saves the data collected in a CSV file, as defined in the configuration file. This data includes the time at which the cycle ended, the measured size of the plant, the measured gross weight and the weight converted into grams.

This means that the station no longer require to be connected to the database at all time. However, this lead to some limitations:

- Some data may be missing from the database if it could not be sent at the end of the cycle.
- As the stations update the date and time by checking for the correct date and time via the Internet when they boot up (and once per day), if they are switched off for a period of time and not connected to the Internet when they boot up again, this can lead to inconsistencies in the dates recorded for data acquisition.

However, this change allows the stations to operate without being connected to an access point. The CSV file must be retrieved manually, either via a remote connection or directly from the device's memory card. A guide to connect via Secure Shell (SSH) is provided at the end of the `README.md` file, available in Appendix F and on the project repository.

Additionally, the images captured by the station are stored in the data/images folder (by default), which enables the dataset captured by the station to be collected in case of an image processing problem or as a new dataset to test new image processing methods.

Moreover, the current status of the station and details of the last encountered exception are sent to the database with the measurements, as previously described in section 6.4. This enables the possibility to check the status of each station remotely from the Grafana interface.

Finally, the last image is converted into a string (sequence of characters) and sent to the database. This is done by reading the image as a binary file, which is then encoded using a base-64 encoding to convert the binary data into a printable American Standard Code for Information Interchange (ASCII) string format. This works by representing the binary data using a set of 64 characters (A-Z, a-z, 0-9, + and /). The resulting string can then be sent to InfluxDB and decoded and displayed in Grafana. This makes it possible to remotely check if the stations are correctly positioned or if they need to be readjusted.

Table 6.1 gives an overview of the data sent to the database before and after the modifications. It is important to note that for the conversion in grams to be accurate, the load cell must be calibrated prior to starting the station using the `calibration.py` Python script located in the `tool/` folder. The details of this script are explained in chapter 5.

Field	Before	Now
Measured weight	Yes	Yes
Weight in grams	No	Yes
Growth	Yes	Yes
Last picture	No	Yes
Station status	No	Yes
Error encountered	No	Yes

Table 6.1: Different data sent to InfluxDB

For reference, an image taken by the station is on average 27227 bytes, and a measurement line in the CSV files is on average 74 bytes. Once fully set up on a 16 GB SD card, there is still more than 9 gigabytes available on the system. This means that a station can store up to this before running out of space:

$$\frac{9 * 10^9 \text{ bytes}}{27227 + 74 \text{ bytes}} = 329658.252555 \approx 329658 \text{ measurements}$$

This implies that if a new measurement is taken every 60 seconds, as is currently configured, a station can theoretically remain offline for 228 days, 22 hours, and 18 minutes before exhausting its available storage space.

Chapter 7

Data processing and visualisation

This chapter will examine the modifications implemented to enhance the stability of weight collection utilising the scale device, in addition to exploring the various possible data graphical representations through Grafana.

7.1 Weight measurement

As previously mentioned in section 3.4.4, the HX711 controller frequently returns anomalous readings that may be caused by several factors. The most probable causes may be attributed to electrical noise, interferences from other nearby devices or environmental factors such as high humidity or temperature variations. As a result, the weight measurement is rendered unreliable.

7.1.1 Data post-processing

Although the data collection may be unreliable, it is still possible to remove any outliers or anomalous readings by processing the data at a later stage, thus ensuring the integrity of the final result. Figure 7.1 depicts a potential data cleansing process that employs a rolling median. A rolling metric is defined as a calculation that represents the evolution of values by aggregating those within a specified window. In this instance, the median was employed to aggregate values over the preceding seven occurrences, thus providing a smoothed representation of the measured weight of the plant over time.

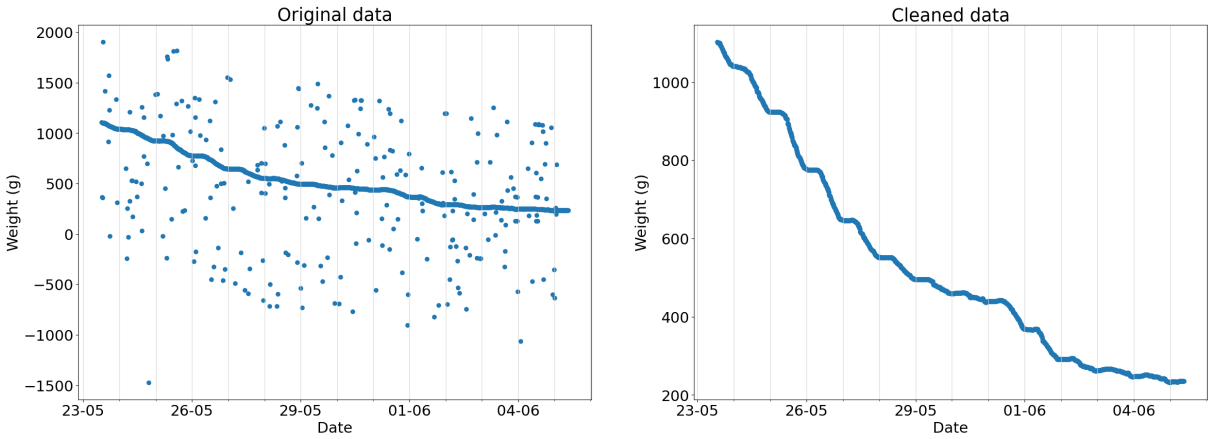

Figure 7.1: Data cleaning using a rolling median (window = 7)

However, this data cleaning process can only be performed after the measurements have been collected, and it does not prevent erroneous readings from occurring. Implementing a method to eliminate these anomalous readings at the time of collection would be a preferable solution, as it would ensure the accuracy and reliability of the measurements, as well as directly providing clean data to the students and teaching assistants using the stations.

7.1.2 Filter data returned by the HX711 converter

In order to implement a solution that enables the reliable measurement of the weight data from the HX711 converter without the inclusion of outliers, this experiment was conducted with the objective of evaluating the efficacy of various statistical filters on data sets of varying sizes taken at the time of measurement.

Materials and methods

To conduct this experiment, a prototype station was placed in a greenhouse. A pot containing a week-old *Zea Mays* plant was placed on the balance connected to the station. The data acquisition is managed by a Raspberry Pi Zero W connected to the HX711 converter. A set of 1,000 measurements is collected every 10 minutes. The sets of data are saved in a CSV file at the end of each measurement cycle.

Results

As anticipated, the values returned from the HX711 controller varied considerably. Figure 7.2 shows the first 10 values collected in each set of measurements.

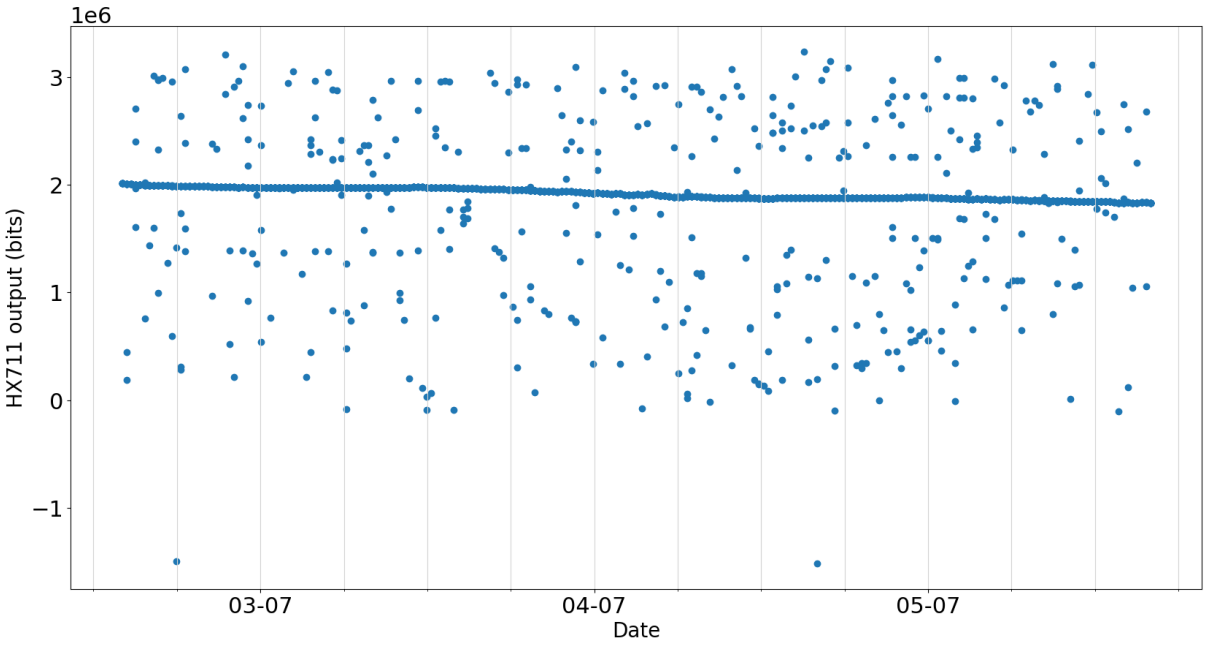

Figure 7.2: Raw output of the HX711 controller

However, it is possible to eliminate the outliers by aggregating each set of measurements using different methods. Three methods were tested on the collected sets of measurements. The methods used were:

- Computing the mean value of the whole dataset
- Computing the median value of the whole dataset
- Computing the mean value of the data within the Interquartile Range (IQR), which are the values within the 25th (Q1) and 75th (Q3) percentile of the data

Figure 7.3 illustrates the results obtained from the various methods applied to sets of 1,000 data points per measurement. As can be seen, simply using the mean value is not an adequate method, as the anomalous data were significant enough to influence the remainder of the dataset. Nevertheless, the median value and the median of the interquartile range provide very close results, without the influence of the outliers.

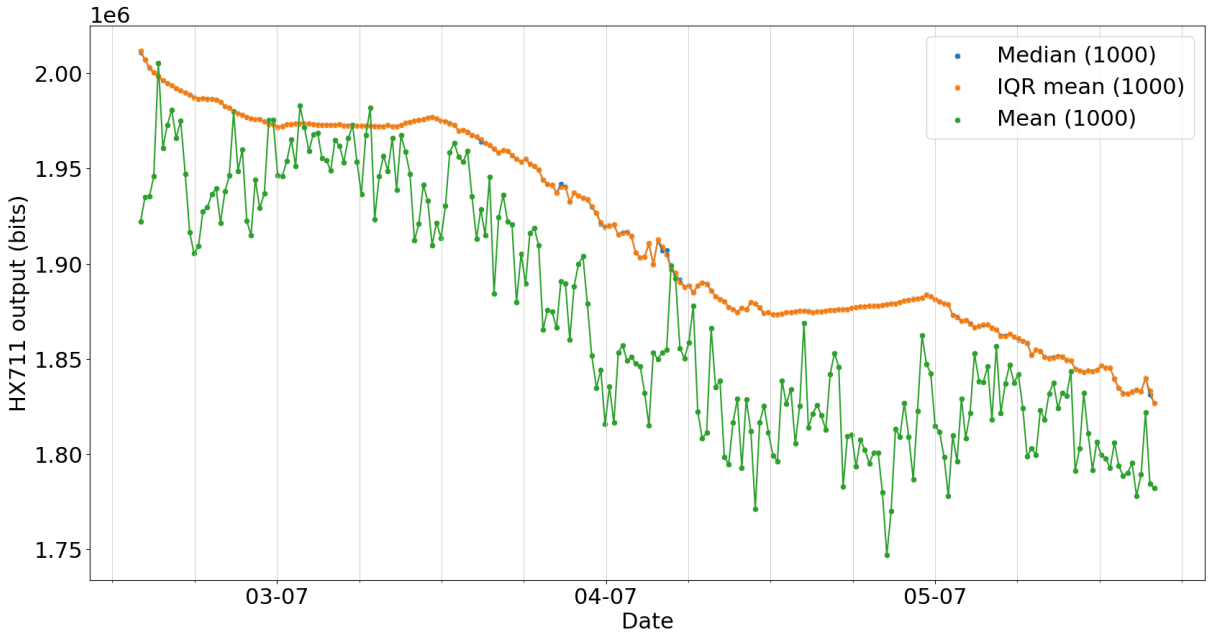

Figure 7.3: Comparison between the different aggregation methods on the entire dataset

It is important to note that the process of collecting 1,000 measurements takes, on average, approximately 680 seconds, or 11 minutes, with an average time for a single measurement of 0.7 seconds. This is a potential issue, as having the first data point from the collected set more than 10 minutes after the last one introduces the potential for significant differences between the two points due to changes in the environment.

To resolve this issue, the median value and the median of the interquartile range methods were tested on smaller data sets of 10, 25, 50 and 100 measurements. This was done to ascertain whether any notable differences could be distinguished and to determine whether utilising these methods on smaller samples could be a viable approach. The results are illustrated in Figure 7.4.

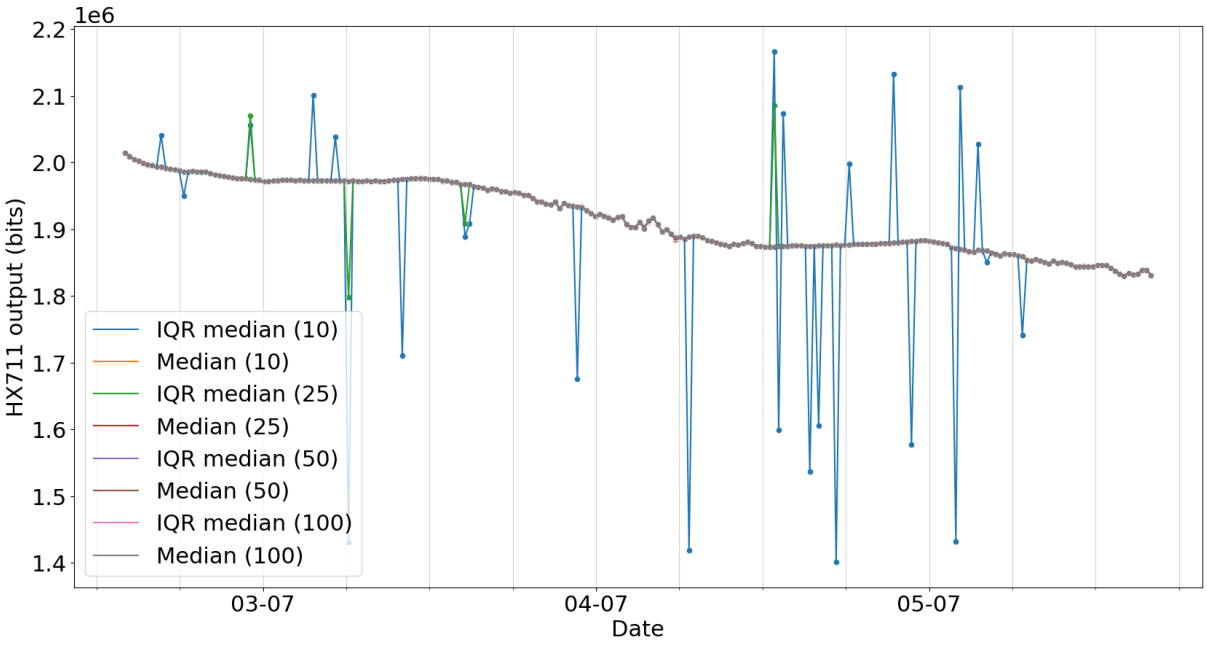

Figure 7.4: Comparison between the different aggregation methods on smaller data sets

As can be seen, although the interquartile range eliminates outliers, the data may still be sufficiently different to influence the mean of the remaining data. Therefore, this method requires a relatively large set of data to provide a stable and reliable solution. In contrast, when using the median, it provides stable data with only a set of 10 measurements.

Conclusion

To guarantee the accuracy of the measurements, it is vital to gather a sufficient number of data points and employ the median value to represent the weight. However, it is crucial to ensure that the measurements are taken at close intervals to maintain their consistency. To this end, the method implemented consists of collecting a set of 10 readings from the HX711 controller and utilising the median of this set as the definitive value. Figure 7.5 illustrates the result of applying this method to the data set comprising 10 data points per measurement on the left-hand graph, with the resulting graph on the right.

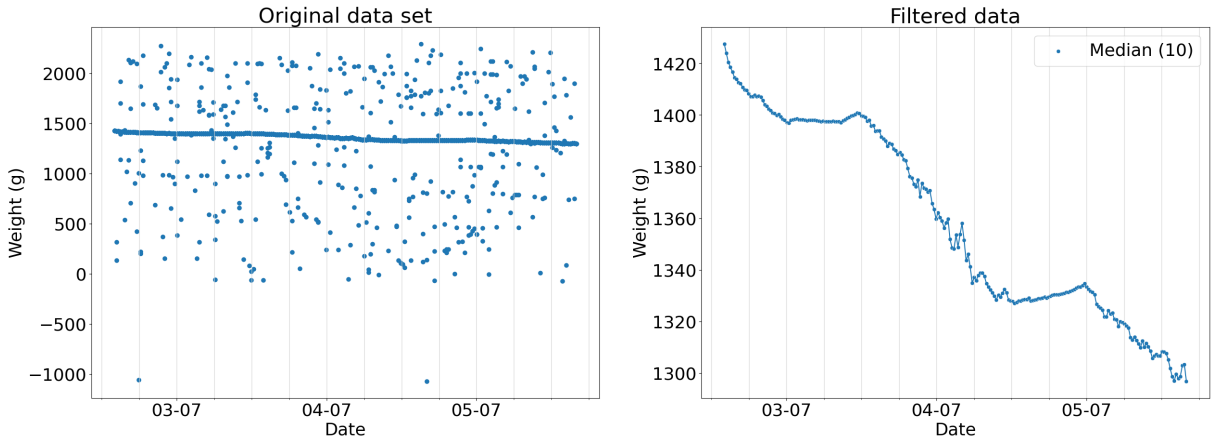

Figure 7.5: Data collection using the median of a set of 10 readings

7.2 Data visualisation

As previously stated in Section 6.6, a number of alterations were made to the data sent to InfluxDB, resulting in changes to the information viewable in Grafana. Figure 7.6 illustrates an example of a Grafana dashboard displaying the diverse metrics available for each station.

Figure 7.6: Example of a Grafana dashboard for multiple stations

As demonstrated, it is possible to monitor the data transmitted by each station from a single dashboard. Moreover, the addition of the status of each station, which in Figure 7.6 are visible in the table in the lower left corner, facilitates the monitoring of each station from a single point.

Additionally, it is possible to create distinct dashboards for each station, thereby concentrating the data associated with a specific station. When focusing on a single station, it is also possible to display the last picture taken by the station by converting the image from a base 64 string back to an image format. It should be noted that this functionality is only available if the *Business Media* plugin has been added to Grafana, as this plugin provides panels that are able to display Base64 encoded images. Figure 3 illustrates an example of a dashboard displaying the data sent by a single station.

Figure 7.7: Example of a Grafana dashboard for a single station

Furthermore, Grafana provides the option of creating new users with varying permissions within the organization. This is useful in our case, as it allows the creation of a new user for each group of students, who will then be able to view only the dashboard that they have access to. For instance, a user named “Group 1” can be created and configured with permissions that allow them to only access the dashboard linked to Station 1. Appendix E.2.1 provides instructions on how to create dashboard, user, and set permissions in Grafana.

Chapter 8

Future work

This chapter will examine potential areas for future improvement. First, it will address the current constraints of image analysis, which has not been previously examined in depth in this thesis. Next, it will present modifications to the hardware components of the station. Finally, it will conclude with a discussion of changes that could be made to improve the design of the stations.

8.1 Image processing

The current state of image analysis is somewhat limited in terms of its capabilities, with results that are highly variable. Indeed, the temporal instability of illumination renders the automatic analysis of images using colour channels alone impractical. This instability means that a threshold used to create a mask of the plant and use it to create its skeleton will be valid for one image but not necessarily for another. Figure 8.1 illustrates two different results given by the binary thresholding process using the same value but with different lighting and background noise.

Figure 8.1: Different result of binary images created gray images based on the same threshold value

To address this issue, Martin Lallemand and Louis Lemaire employed the Canny filter method to identify the contours of the maize and subsequently reconstructed the plant's shape through the application of a morphological closure operation, as discussed in section 3.4.4 [1]. However, further research could be conducted to explore alternative methods for improving the stability of the image processing pipeline.

A possible initial approach could be to white balance the images captured in order to address discrepancies between images resulting from variations in lighting. This could facilitate the uniformity of image processing steps, such as thresholding, across different images. Furthermore, it may be of interest to utilise this method with different background colours in order to facilitate the isolation of the plant from the background and any potential shadows.

Additionally, it would be advantageous to examine the alternative methodologies offered by the PlantCV library. Indeed, the PlantCV library provides a comprehensive range of methods and tools for image analysis, including automatic thresholding techniques. The utilisation of these with a standardised background could potentially yield more effective results than the current method.

Nevertheless, it is important to note that the current station hardware is no longer compatible with the latest version of PlantCV. Consequently, it is essential to verify whether the specifications of certain functions have been modified between the version utilised by the station (v3.14.3) and the latest version when evaluating other tools and algorithms with a more recent version of PlantCV.

8.2 Hardware components

As mentioned above, the Raspberry device currently utilised by the station is a Raspberry Pi Zero W, which is based on a 32-bit architecture. This can potentially lead to compatibility

issues, as some software may require more memory address space than is available from a 32-bit architecture. This is the case for image processing libraries such as PlantCV, whose version v4.0 and later has become incompatible with 32-bit systems.

In order to fully utilise the capabilities of the library and take advantage of its latest features, it would be advantageous to replace the Raspberry Pi Zero W with a Raspberry Pi Zero 2 W, which is based on a 64-bit architecture. Furthermore, its quad-core processor could be employed to multithread the measurement pipeline of the station, enhancing its overall performance while only representing a 3 euro increase from the Raspberry Pi Zero W [46].

Another aspect to consider the internal cable configuration within the case, which is suboptimal. The various components are connected using simple Dupont cables, which is not an adequate solution for a system that might be manipulated frequently. Indeed, during the experiments conducted this year, it occurred on several occasions that a cable disconnected, which resulted in the station being unable to properly communicate with its components. It may therefore be beneficial to consider the design of a printed circuit board (PCB) that incorporates designated areas for soldering the various components. This approach could accelerate the development of a station, minimise production errors and enhance its reliability.

8.3 Station design

Considering the design of the station, it would be advantageous to design and print a custom-made box in 3D that would be less cumbersome while still providing sufficient space for the various components of the station. Additionally, the current design requires the drilling of multiple holes within the box to accommodate the cables of the LED lightning strip, the buttons, and the screen, which is a rather inefficient approach.

It would also be beneficial to identify a solution that would enable the Picamera to be mounted on an adjustable base, thus facilitating its positioning at the optimal height. At present, it is necessary to elevate the entire station in order to position the camera at the correct height, which has the dual disadvantage of reducing the station's stability and increasing the risk of it falling over, particularly if multiple students are present in the room.

Chapter 9

Conclusion

Louis Lemaire and Martin Lallemand have proposed an innovative and interactive approach to practical exercises in plant physiology. The prototype low-cost phenotyping station that they have produced represents an interesting and promising platform for this new approach, which aims to involve students more closely in practical sessions.

The prototypes, as received in September, provide a robust foundation for development of a final solution, despite inherent limitations in their design and the current lack of robustness in their implementation.

The first step undertaken was to change the chosen operating system for a lighter alternative. Doing so will limit the amount of resources wasted by processes that are not necessary for this project. The aim of making this change is to ensure that the stations have the resources they need to function properly, despite the limited capacity of their hardware.

The discussion then focused on the proposal of several mechanisms for the automation of the installation and configuration of a station. This is with the objective of reducing the time required for the setup and configuration of a station, thus facilitating the setup of multiple stations.

In order to enhance the robustness, modularity and maintainability of the software, its implementation has been reworked. The utilisation of diverse mechanisms, including the incorporation of both internal and external documentation, the implementation of a logging and error management system, and a refactoring of the implementation, has enabled the establishment of a more robust, modular and user-friendly foundation for future developments.

The measurement of transpiration via weight measurement on the scale device has also been modified to yield more accurate results. This was achieved by implementing a filter at the time of data collection, which serves to enhance stability and remove as many outliers as possible. Furthermore, the display of data from each station has been redesigned to

facilitate centralized monitoring of the status of each one, while also allowing for separation between users so that each group of students can access the data from their respective station directly.

These modifications have been implemented with the objective of consolidating and enhancing the work pioneered by Louis Lemaire and Martin Lallemand. This project represents not only an innovative challenge but also a promising avenue for further research.

Nevertheless, several constraints remain before this prototype can be considered a final product. Image analysis has yielded promising results, but still faces numerous constraints that must be addressed to render it viable. Similarly, the prototype's design and hardware have demonstrated potential for improvement. In particular, the fabrication of a PCB could enhance the prototype's robustness, facilitating the design of a finished product suitable for direct use by students in practical work.

Appendix A

DietPi installation

A.1 Write the latest DietPi version to a SD card

The latest image of the latest version of DietPi can be downloaded from the official website[32]. Upon accessing the download page, select the appropriate image corresponding to the hardware in use, in this case the image for Raspberry Pi Zero (1). Figure A.1 illustrates the steps for downloading the image from the website.

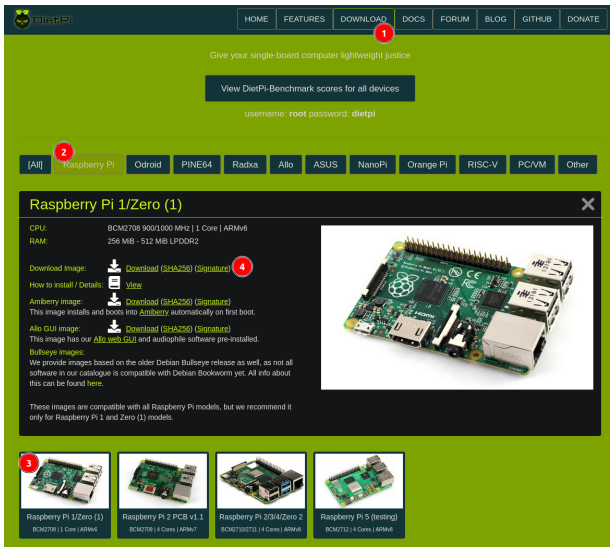

Figure A.1: Download the latest image of DietPi

Once the download is complete, the image must be can be written to a SD Card using a disk imaging tools. Here, we will use balenaEtcher[47] which is free, open-source, and

available on Linux, Windows and macOS. It can be downloaded directly from their official website.

Once balenaEtcher has been installed and is operational, the user is required to select the "Flash from file" option (step 1) and navigate to the previously downloaded DietPi image (step 2). Once the image has been loaded, it is then necessary to select the target disc on which the image is to be written (steps 4, 5 and 6). It is recommended that an SD card of at least 16 GB be used for this project. Once the image has been loaded and the appropriate target drive has been selected, the process of flashing the image to the SD card can be initiated by selecting the "Flash!" option (step 7). balenaEtcher will then request administrator access in order to overwrite the data stored on the target drive, and will then proceed to flash the image. Once the process is complete, the SD card, keyboard, and screen can be inserted into the Raspberry Pi Zero W and the device powered on to proceed with the system configuration.

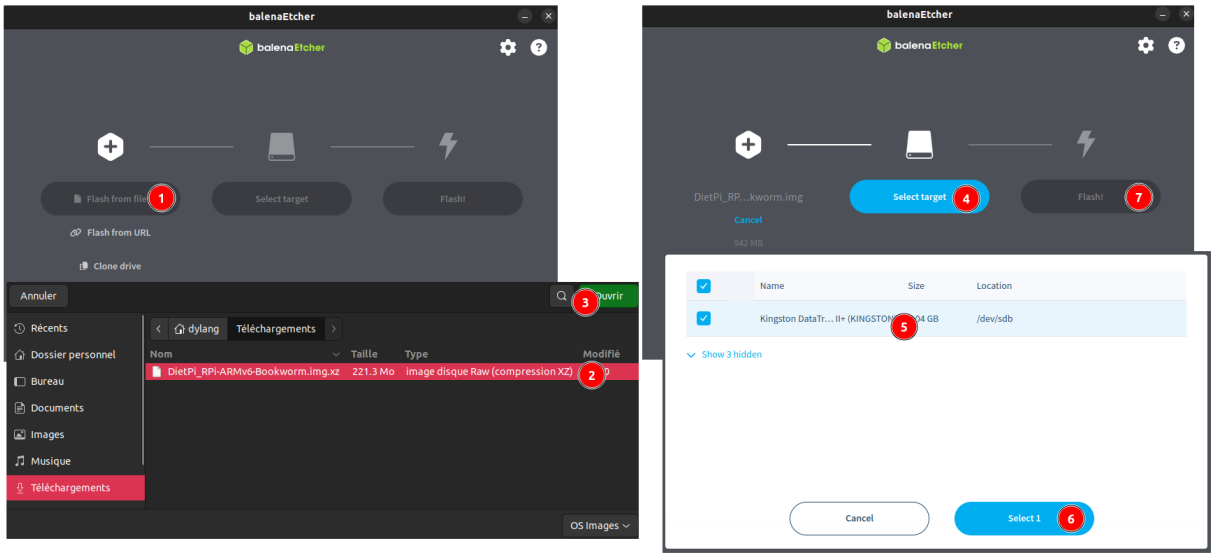

Figure A.2: Steps to flash a SD card using balenaEtcher

A.2 Connection via a remote SSH connection

1. Make sure that you are connected to the same network as the Raspberry Pi.
2. Find your current IP using 'ifconfig /all' (on Windows or macOS) or 'ip a' (on Linux)
   - You should see an IP address in the form of '192.168.1.X', '10.42.0.X' or similar.

```
PS C:\Users\goffi> ipconfig /all
[...]
Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : home
    Description . . . . . . . . . . . : Killer(R) Wi-Fi 6 AX1650i 160MHz Wireless
    ↪  Network Adapter (201NGW)
    Physical Address. . . . . . . . . : 70-A8-D3-1B-15-93
    DHCP Enabled. . . . . . . . . . . : Yes
    Autoconfiguration Enabled . . . . : Yes
    IPv6 Address. . . . . . . . . . . :
    ↪  2a02:a03f:c09a:cf01:6a01:a243:a90:2561(Preferred)
    Temporary IPv6 Address. . . . . . :
    ↪  2a02:a03f:c09a:cf01:9dad:a85e:25f2:bb1e(Preferred)
    Link-local IPv6 Address . . . . . : fe80::4101:1239:213e:be14%2(Preferred)
    IPv4 Address. . . . . . . . . . . : 192.168.129.99(Preferred)
    Subnet Mask . . . . . . . . . . . : 255.255.254.0
    Lease Obtained. . . . . . . . . . : 17 July 2024 09:40:54
    Lease Expires . . . . . . . . . . : 17 July 2024 10:40:52
    Default Gateway . . . . . . . . . : fe80::22b8:2bff:fe0b:9f62%2
                                        192.168.166.254
    DHCP Server . . . . . . . . . . . : 192.168.166.254
    DHCPv6 IAID . . . . . . . . . . . : 40937683
    DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2A-30-89-94-08-8F-C3-64-23-E1
    DNS Servers . . . . . . . . . . . : fe80::22b8:2bff:fe0b:9f62%2
                                        192.168.166.254
    NetBIOS over Tcpip. . . . . . . . : Enabled
```

Figure A.3: Example output of `ifconfig /all`

- If your IP is, for example, `192.168.34.17`, then your IP range is `192.168.34.0/24`. In the above example, the IP is `192.168.166.254`, so the IP range is `192.168.166.0/24`.

3. Find the IP address of the Raspberry Pi:
   - Using `nmap`, scan the network using 'nmap -sn <YOUR_IP_RANGE>' (replace '`<YOUR_IP_RANGE>`' with your IP range found at step 2).
   - The Raspberry Pi should appear in the list with "Raspberry Pi Foundation" next to the MAC Address, its IP will be just above.

```
PS C:\Users\goffi> nmap -sn 192.168.166.0/24
Starting Nmap 7.95 ( https://nmap.org ) at 2024-07-17 09:47 Romance Summer Time
Nmap scan report for 192.168.166.16
Host is up (0.013s latency).
MAC Address: B8:27:EB:BF:6D:1A (Raspberry Pi Foundation)
Nmap scan report for 192.168.166.254
Host is up (0.017s latency).
MAC Address: 4E:A5:88:52:12:F3 (Unknown)
Nmap scan report for 192.168.166.174
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.92 seconds
```

Figure A.4: Example output of `nmap -sn`

- On A.4, the Raspberry Pi has the IP `192.168.166.16` (the first in the list).
- It should be noted that the use of the `nmap -sn` command to scan a network will create a load on the network, which could be misinterpreted as the discovery phase of an attack. Thus, it is strongly recommended that this method be employed with caution and only in controlled or private networks.

4. Connect to the Raspberry Pi using SSH with `ssh root@<RASPBERRY_PI_IP>` (replace `<RASPBERRY_PI_IP>` with the IP found at step 3). Enter the password when prompted. On a default DietPi configuration, the password is `dietpi`.
5. You can close the connection at any time using `exit`.

Appendix B

Pricing and source of each station component

The Table B.1 presents the hardware components that compose the station, with their associated prices and sources. Although the components remain the same as those proposed for use in the prototypes, some prices and sources were no longer available. Consequently, they have been updated with up-to-date information.

As the price of Raspberry Pi computers varies according to availability, the availability, and pricing information can be found on the *rpilocator.com* website [48].

Component	Quantity	Price (€)	Source
Raspberry Pi Zero W with headers	1	19.79	MCHobby [1]
Micro USB Power Supply (5V, 3A)	1	2.71	AliExpress [2]
ST7735 LCD screen (128*160)	1	1.90	AliExpress [3]
Button	2	0.19	AliExpress [4]
LED lighting strips	1	1.69	Action [5]
Relay module KY-019	1	0.56	AliExpress [6]
Picamera with Raspberry Pi Zero adapter	1	2.86	AliExpress [7]
Tal226 load cell	1	11.54	AliExpress [8]
HX711 converter	1	1.17	AliExpress [9]
16 GB SD card	1	3.24	AliExpress [10]
WAGO splicing connector with lever (5 connectors)	3	3.3	AliExpress [11]
Scale device (3D printing)	1	5	Equivalent of around 250g of plastic filament
Cable (4 wires) 1.5m	1	1.37	AliExpress [12]
Female to female cable	10–15	0.6	AliExpress [13]
Male to female cable	10–15	0.6	AliExpress
Male to male cable	10–15	0.6	AliExpress
TOTAL		57.12	

Table B.1: Hardware component, costs, and sources for building a PhenoHive station (August 2024)

[1] https://shop.mchobby.be/en/motherboards/1892-raspberry-pi-zero-w-with-header-v11-3232100018921.html

[2] https://www.aliexpress.com/item/1005004113767773.html

[3] https://www.aliexpress.com/item/32843115817.html

[4] https://www.aliexpress.com/item/1005006380840928.html

[5] https://www.action.com/fr-be/p/3011541/ruban-led-usb-maxxter/

[6] https://www.aliexpress.com/item/32727119580.html

[7] https://www.aliexpress.com/item/32901067278.html

[8] https://www.aliexpress.com/item/32691235830.html

[9] https://www.aliexpress.com/item/32883776006.html

[10] https://www.aliexpress.com/item/1005003189934704.html

[11] https://www.aliexpress.com/item/1005003890138184.html

[12] https://www.aliexpress.com/item/2251832867185900.html

[13] https://www.aliexpress.com/item/4000323249974.html

Appendix C

Components wiring

The three five-input WAGO connection terminals facilitate the powering of the various components by connecting several cables to a single input. Given the limited number of power supply pins on the Raspberry Pi Zero W, specifically 5 volts, 3.3 volts and GROUND, the WAGO terminals serve to expand the number of available power supplies. Table C.1 provides a listing of the connections for each component to the Raspberry Pi pins, and Figure C.1 presents an electronic diagram of the station.

ST7735 LCD screen	PIN	GPIO
LED	3.3V	3.3V
SCK	23	11
SDA	19	10
A0	18	24
RESET	22	25
CS	24	8
GND	GND	GND
VCC	3.3V	3.3V
HX711 controller	PIN	GPIO
VCC	3.3V	3.3V
SCK	31	6
DT	29	5
GND	GND	GND
Relay Module KY-019	PIN	GPIO
DATA	16	23
VCC	5V	5V
GND	GND	GND
Left button	PIN	GPIO
DATA	40	21
GND	GND	GND
Right button	PIN	GPIO
DATA	36	16
GND	GND	GND

Table C.1: Components wiring to the Raspberry Pi Zero W

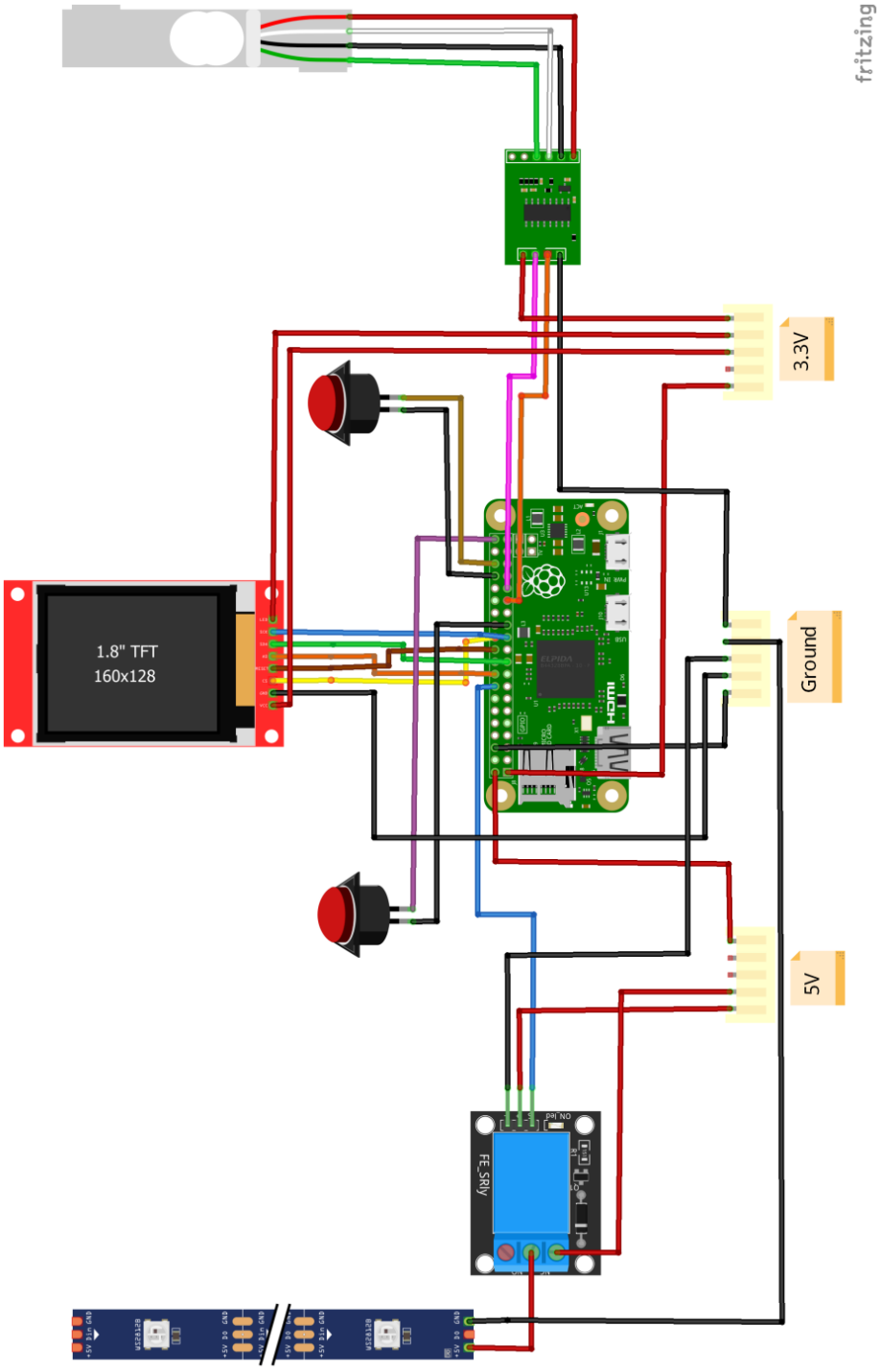

Figure C.1: Electronic diagram of a station [1]

Appendix D

Configuring a Raspberry Pi as access point

The following steps are required to configure a Raspberry Pi running on Raspberry Pi OS Desktop as an access point [49].

It should be noted that for the Raspberry Pi to be able to share its internet connection, it must be connected via Ethernet to a router.

Once the Raspberry Pi has been configured with a version of the Raspberry Pi OS Desktop, the next step is to connect it to a keyboard, mouse, and display and then open a terminal.

First make sure that the system is up-to-date using the following command:

```
sudo apt-get update && sudo apt-get upgrade -y
```

Next, open the raspi-config software using the following command:

```
sudo raspi-config
```

Once the configuration software is open, navigate to the *Advanced Options* menu, then enter the *Network Config* menu and select enable the *Network Manager*. Once this is done, exit the software using the *Finish* option and select *Yes* to reboot the system and apply the changes.

Wait for the Raspberry to reboot, then open the wireless options by clicking the network icon in the upper right corner of the screen, select *Advanced Options* and then *Create Wireless Hotspot*.

A prompt will appear on the screen. The user is required to enter the name of the network to which the access point is connected, select the option for *WPA2* as the WiFi security protocol, and enter the password for the access point. Once the information has been

entered, the user must click the button marked “Create” and then reboot the Raspberry Pi. Once it has completed the reboot process, the new WiFi network will be broadcast.

Additionally, the IP address of the device on the new network can be obtained by hovering the cursor over the network icon. This information is essential for configuring the PhenoHive system.

Appendix E

Installation and configuration of InfluxDB and Grafana

E.1 InfluxDB

The following guide described the steps required to install and configure InfluxDB V2 on a Raspberry Pi [50].

First, the repository containing the InfluxDB package can be added with the following commands:

```
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c
↪  influxdata-archive_compat.key' | sha256sum -c && cat
↪  influxdata-archive_compat.key | gpg --dearmor | sudo tee
↪  /etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg > /dev/null
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg]
↪  https://repos.influxdata.com/debian stable main' | sudo tee
↪  /etc/apt/sources.list.d/influxdata.list
```

Next, the InfluxDB package can be installed using these commands:

```
sudo apt-get update
sudo apt-get install -y influxdb2
```

Once the installation in complete, the service must be enabled and started using the following commands:

```
sudo systemctl enable infludb.service
sudo systemctl start infludb.service
```

Once the service is operational, to access the InfluxDB interface, simply open the web browser on the Raspberry Pi and enter the following URL: `http://localhost:8086/`. Upon accessing the menu, the user is required to create a user account. The necessary information to create an account can be found in Figure E.1.

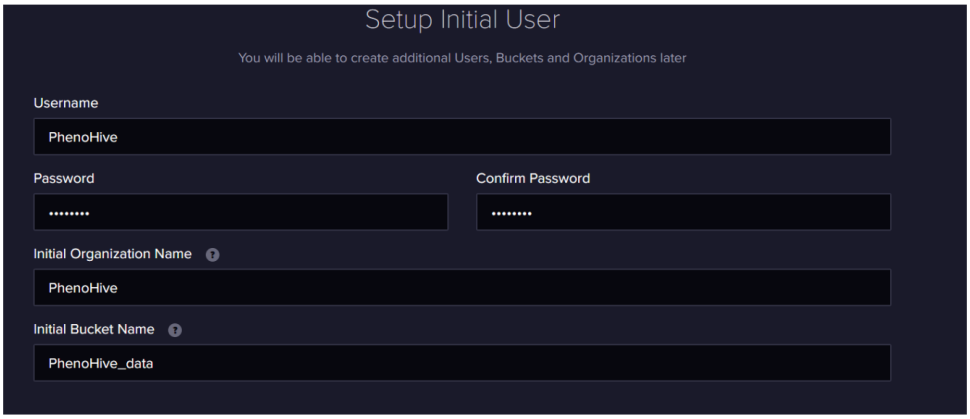

Figure E.1: Setup of the initial user of InfluxDB

These parameters consist of the user identifiers and the organisational identifier, which represents a workspace for multiple users. An organisation may comprise a number of data tables. In this context, the term "bucket" is used to designate a data table. In this instance, the "PhenoHive_data" table will be generated upon the creation of the user, although the creation of additional tables is possible at a later stage.

Once the user has been created, a token is displayed, which enables the connection to the database and the transmission of data. It is important to record this token, as it cannot be accessed a second time. However, it is still possible to create other tokens via the main menu, with different authorisations, or to create a new bucket.

E.2 Grafana

The following guide described the steps required to install and configure Grafana on a Raspberry Pi [51]. It should be installed on the same system as InfluxDB for ease of use.

First, the repository containing the Grafana package can be added with the following commands:

```
wget -O- https://packages.grafana.com/gpg.key | gpg --dearmor | sudo tee
↪ /usr/share/keyrings/grafana-archive-keyring.gpg >/dev/null
echo "deb [signed-by=/usr/share/keyrings/grafana-archive-keyring.gpg]
↪ https://packages.grafana.com/oss/deb stable main" | sudo tee
↪ /etc/apt/sources.list.d/grafana.list
```

Next, the Grafana package can be installed using these commands:

```
sudo apt-get update
sudo apt-get install -y grafana
```

Once the installation in complete, the service must be enabled and started using the following commands:

```
sudo systemctl unmask grafana-server.service
sudo systemctl start grafana-server
sudo systemctl enable grafana-server.service
```

Once the service is operational, to access the InfluxDB interface, simply open the web browser on the Raspberry Pi and enter the following URL: `http://localhost:3000/`. The default credentials are user: *admin* and password: *admin*. They can be changed at any time in the *Administration* panel of the Grafana interface.

To connect InfluxDB to Grafana, once in the main menu, navigate to *Connections*, select *Add new connection* and search for InfluxDB.

In order to connect the database, several parameters needs to be modified:

- The URL of InfluxDB. If it is installed on the same system as Grafana, it should be `http://localhost:8086/`
- The organisation name, here `PhenoHive`
- The token of the bucket
- The bucket name, here `PhenoHive_data`

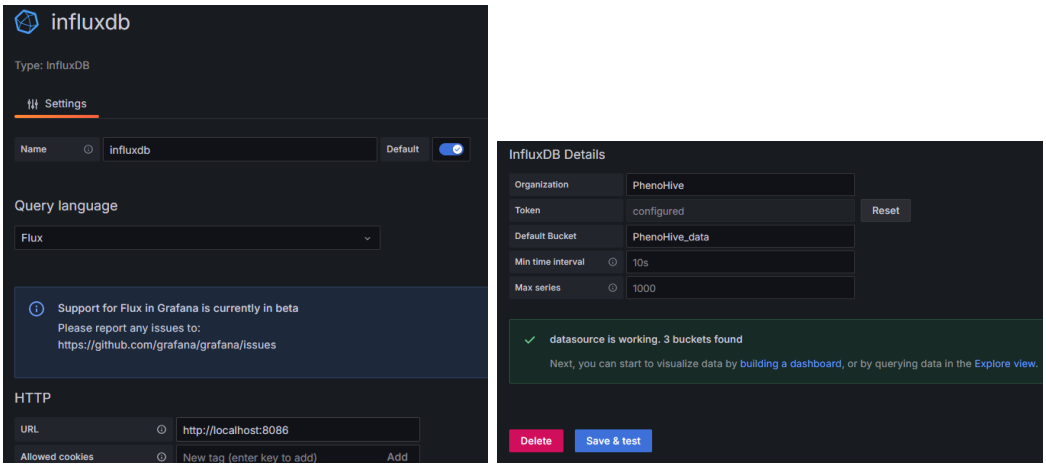

Figure E.2: Linking the InfluxDB database to Grafana

Once the information has been entered, the configuration can be tested using the *Save & test* button. If everything was set up correctly, a green message will appear, as illustrated by Figure E.2

E.2.1 Creating a new dashboard

Once the configuration of Grafana has been completed, the next step is to create a dashboard that will display the data available in InfluxDB. To do so, navigate to *Home* > *Dashboards* and select the *New dashboard* option.

In order to create a new visualisation panel, select the "*+ Add visualisation*" option. This will open a prompt requesting the selection of a data source. Select "*InfluxDB*" as the source and the "*Edit panel*" interface should be open, as illustrated in Figure E.3.

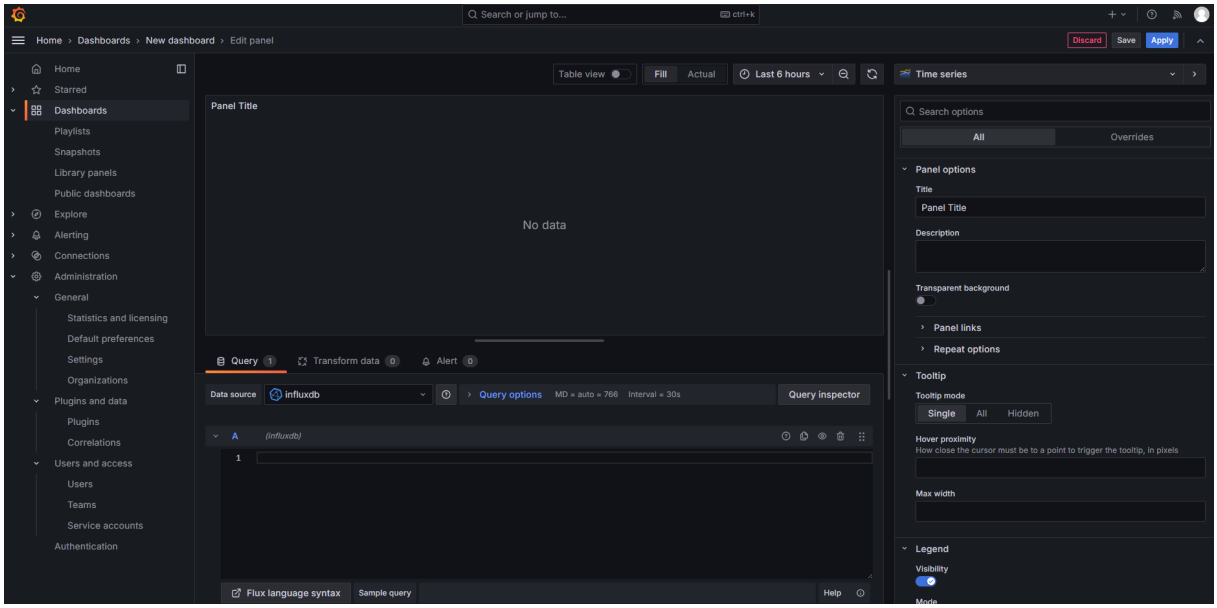

Figure E.3: Creation of a panel in Grafana

On this page, you can enter the InfluxDB query in the "*Query*" panel on the lower centre part, and edit how the data are displayed, the type of visualisation (Time series, Table, Bar Chart, etc.) and other information related to the panel with the menu on the right.

The queries are made using the "*Flux*" [52] language, which was made to query analyse and work on data. For instance, the following query retrieves growth values send by station 1.

```
1 from(bucket: "PhenoHive_data")
2   |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3   |> filter(fn: (r) => r["_measurement"] == "station_1")
4   |> filter(fn: (r) => r["_field"] == "growth")
```

As we can see, it fetches the data from the `PhenoHive_data` bucket, where the measurement is `station_1` and the field is `growth`. As explained in section 6.6, each station stores it data in a different measurement under the format `station_ID` where ID is the ID of the station as defined in its `config.ini` file. Each data collected by the station is stored in a different *field*. The available fields are explained in Table E.1.

Field	Explanation
growth	Growth of the plant in pixels
weight	Weight of the plant in bits
weight_g	Weight of the plant in grams
picture	Last picture taken by the station as a base 64 string
status	Indicator of the status of the station (0 = OK, -1 if encountered an error)
error_time	Time at which happened the last error
error_message	Message of the last error

Table E.1: Different field sent to InfluxDB and their correspondence

Once the editing of the panel is complete, the dashboard can be saved by clicking the "*Save*" button. Alternatively, the "*Apply*" button can be selected to preview the changes to the dashboard.

Display a Base-64 string

In order to display the last picture taken by a station, it is necessary to install an additional plugin in Grafana. To do so, navigate to the *Administration > Plugins and data > Plugins* page. Next, search for the *Business Media* plugin by *Volkov Labs* and install it using the *Install* button.

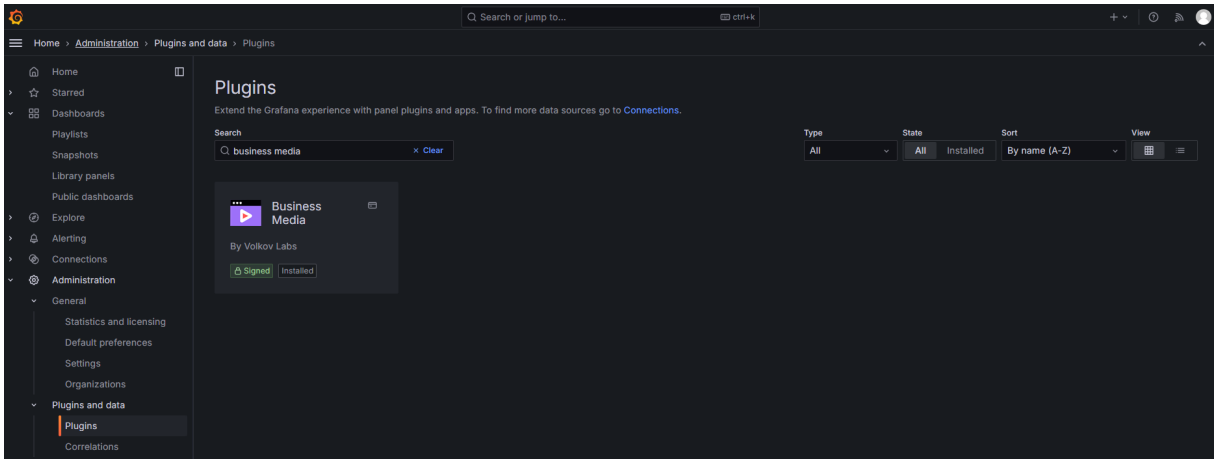

Figure E.4: Grafana — Business Media plugin

Once the plugin has been installed, a new visualisation option called "*Business Media*" will be available when creating a new panel in a dashboard. The *Flux* query to retrieve the image taken by station 1 is the following:

```
from(bucket: "PhenoHive_data")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "station_1")
  |> filter(fn: (r) => r["_field"] == "picture")
```

Once the query as been entered, a message station *Nothing to display...* will appear. This is because it is necessary to precise to Grafana the field where the picture is located. To do so, in the *Business Media* option on the visualisation option on the right, select *Image* as media type and *A:picture* as media field. Once the correct type and field have been selected, apply the changes and the image will appear in the panel, as illustrated by Figure E.5.

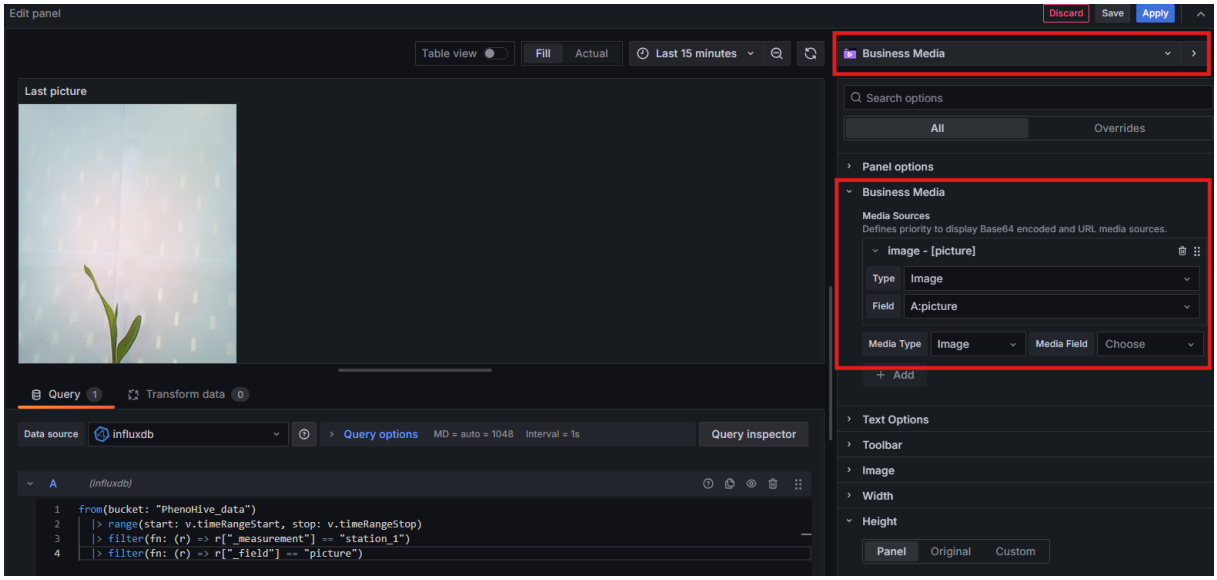

Figure E.5: Display Picture panel in Grafana

E.2.2 Creating new users

It is possible to create a new user within an organisation within Grafana, with the option of assigning different permission levels. To do so, navigate to the *Administration* > *User and access* > *Users* page. This page will display the current users and allow for the creating of new users using the *New user* button. In order to create a new user, it is necessary to enter the desired name, username, and password (which may be modified at any time) into the form and select "*Create user*".

By default, new users will have the "Viewer" role, which only allows them to view the existing dashboard, but not modify them. These permissions can be changed at any time from the *Users* page.

Moreover, it is possible to set permissions for each dashboard that is created. This can be achieved by accessing the "*Dashboard settings*" and navigating to the "*Permissions*" tab. In this tab, it is possible to add, remove and change the permissions of the different roles within the organisation and of its users. Figure E.6 provides an example of dashboard permissions that allow only the user "*group 1*" and the administrator to view the dashboard.

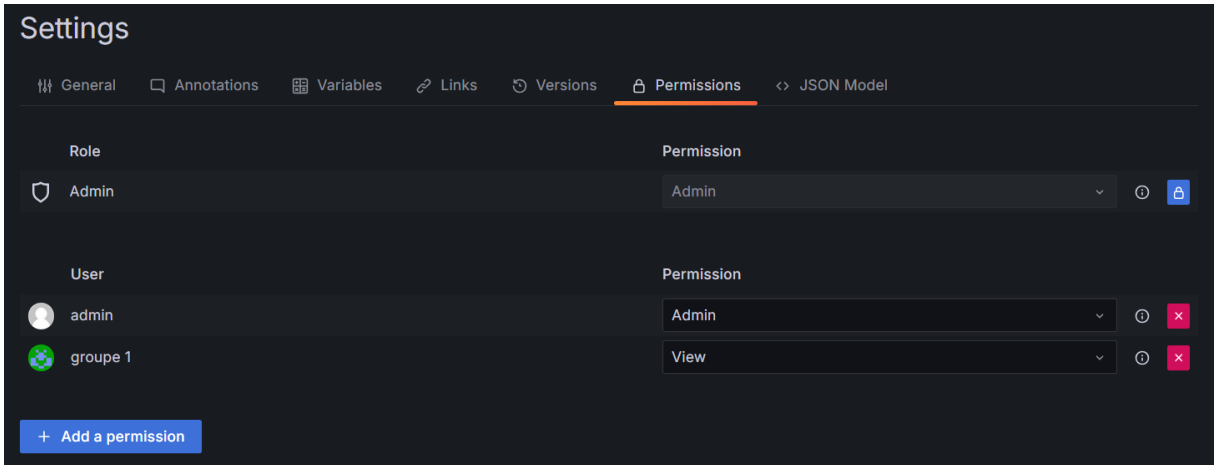

Figure E.6: Grafana dashboard permissions

Appendix F

README.md

This appendix contains a rendered version of the Markdown `README.md` documentation file available on in the root directory of the GitHub repository.

F.1 PhenoHive

Low-cost Raspberry pi-based phenotyping station. Based on a prototype[1] by M. Lallemand and L. Lemaire

F.2 Table of contents

[1] `https://github.com/marty12342000/PhenoHive`

- Project setup
  - Automated setup
  - Manual setup
- SSH connection

F.3 Project Description

PhenoHive is a low-cost station for plant phenotyping.

The design is based on a Raspberry Pi Zero W running on DietPi OS with:

- A Tal226 load cell connected to a HX711 controller to measure plant weight.
- A Raspberry Pi Camera and a LED lightning strip connected to a KY-019 relay module to take pictures of the plant.
- A ST7735 LCD screen and two buttons to interact with the user.

The software is written in Python, with a bash setup script[2] to set up the system. The Python code is divided in five files:

- `main.py` is the main file, it initialises the system and handles the user interactions as well as the different pipelines.
- `PhenoHiveStation.py` contains a singleton class that handles the hardware interactions. It contains the different variables and methods to take pictures, measure weight, and communicate with the database.
- `image_processing.py` contains the different functions to analyse the plant images and compute its growth.
- `show_display.py` contains the Display class, containing the different methods to display the information on the LCD screen.
- `utils.py` contains two functions, one to set up the logger used by the system, and one to compute the growth of the plant.

F.4 System Operation

F.4.1 Configuration

Each variable of the station, such as the different pins of each sensor, the time interval between each measurement, etc. is set in `config.ini`.

[2] `setup.sh`

F.4.2 Initialisation

Once the system has been set up (see Installation for more details), `main.py` will be run at startup. It will initialise the logger, instantiate the PhenoHiveStation class, and start the main loop. At this point, it will display the main menu on the LCD screen and either:

- Wait for the user to press "Start" to enter measurement mode or "Config" to configure enter the configuration menu.
- Automatically start the measurement pipeline if the system unexpectedly shut down in measurement mode.

F.4.3 Configuration Menu

The configuration menu allows the user to:

- Tare the load cell in the "Calib" menu.
- Ensure that the camera is correctly positioned in the "Prev" menu (this menu can be exited by pressing the right button).

F.4.4 Measurement Mode

Measurement pipeline

The measurement mode is divided in several pipelines to improve modularity and ease of use:

- the picture pipeline takes a picture of the plant, saves it `data/images`, and displays it on the LCD screen. Then, the picture is analysed using plantcv to compute the growth of the plant (see `image_processing.py`).
- the weight pipeline measures the weight of the plant, by taking the median of several measurements to avoid abnormal values.
- the database pipeline sends the different measurements to the InfluxDB database. The measurements are also saved in a CSV file in the `data` folder to avoid data loss in case of database failure.

Display and status

The different steps of the pipelines are displayed on the LCD screen to inform the user of the system status. A status menu is also available when the system is not in a pipeline to display the station's status. Furthermore, the display will show a colour outline at all times to inform the user of the system status: - Green: the system is nominal.

- Blue: the system is nominal but not connected to the database.

- Yellow: the system is in a pipeline.
- Red: the system encountered an error. The error message and time will be displayed on the status menu.

Measurement format

The different data sent to the database and saved in the CSV file are: - "weight": the weight of the plant (raw value without a conversion to grams).

- "weight_g": the weight of the plant (in grams if the calibration coefficient was set using `tools/calibration.py`).
- "standard_deviation": the standard deviation of the weight measurements.
- "growth": the growth of the plant (in pixels).
- "picture": the picture of the plant (in base64 format).
- "status": the status of the station.
- "error_time": the time of the last error.
- "error_message": the last error message.

F.4.5 Logging and error handling

The system logs are saved in `logs` folder, with the time and date of the log as the filename. If the logging level is not given as argument when starting the station, the default level is DEBUG.

When an error occurs, the system will register the error message and time using the `register_error` method of the PhenoHiveStation class. The error message will be logged, displayed on the LCD screen, and the status will be set to red. Critical steps, such as when collecting the weight or taking a picture, will be wrapped in a try/except block to catch any error and register it. However, unexpected errors can still occur. In this case, the system will try to catch and register the error, but if more than 10 unexpected errors are encountered, the system will raise a RuntimeError and restart (if the phenohive.service[3] is set to restart on failure).

[3] `tools/phenohive.service`

F.5 Installation

The system is designed to run on a Raspberry Pi Zero W with DietPi OS. The system was tested with DietPi v9.6.1 (released in 2024-07-07).

To install the system, you can either:

- Use the provided pre-built image of the system with the system already installed and configured.
- Follow the steps below to install the system on a Raspberry Pi Zero W with DietPi OS (or other debian-based OS).

F.5.1 Operating System

The steps below are for DietPi OS, but the automated setup script[4] can be used on other Debian-based OS (e.g. Raspberry Pi OS). If you run into any trouble or if you are trying to install on another OS, the steps to set up the project manually are given in the manual setup section.

Using the preconfigured image

1. Download the preconfigured image from here[5].
2. Flash the image on a microSD card of at least 8Gb using Balena Etcher[6].
3. Insert the microSD card in the Raspberry Pi Zero W and power it on.
4. Connect to the Raspberry Pi Zero W using a keyboard and a screen or via SSH (default login: root, password: phenohive).

- The Raspberry Pi Zero W will automatically connect to the internet using the Wi-Fi network configured in `tools/dietpi/dietpi-wifi.txt` file (default: SSID=PhenoHive, Password=Phenohive).
- For a step-by-step guide on how to connect to the Raspberry Pi Zero W using SSH, see below.

5. Navigate to the PhenoHive folder using `cd /root/PhenoHive`, and ensure that the system is up to date using `git pull`.
6. Modify the configuration file `config.ini` according to your needs (see Configuration).
7. Run the calibration script[7] using `python3 tools/calibration.py` to calibrate the load cell (warning: this will overwrite the previous calibration coefficient).

[4] `setup.sh`

[5] `https://uclouvain-my.sharepoint.com/:u:/g/personal/dylan_goffinet_student_uclouvain_be/Eb_mK8L4GptGhppHbfrdEPoBmDWEppfjG-rZpERCGurvsw?e=2BCUSQ`

[6] `https://www.balena.io/etcher/`

[7] `tools/calibration.py`

8. Enable the service using `systemctl enable phenohive.service`, and start the service using `systemctl start phenohive.service`.
9. The system is now running; you can check the status of the service using `systemctl status phenohive.service`.

Automated installation

1. Download the latest version of DietPi OS from the official website[8].
2. Flash the image on a microSD card using Balena Etcher[9].
3. You should have two partitions on the microSD card, one named "bootfs" and the other named "rootfs".
4. Copy the files located in the `tools/dietpi` folder to the "bootfs" partition. A detailed breakdown of the files is given in DietPi_files.md[10].
5. Modify the dietpi-wifi.txt[11] file to include your Wi-Fi network SSID and password (default: SSID=PhenoHive, Password=Phenohive).
6. Insert the microSD card in the Raspberry Pi Zero W and power it on.
7. Connect to the Raspberry Pi Zero W using a keyboard and a screen or via SSH (default login: root, password: phenohive).

- The Raspberry Pi Zero W will automatically connect to the internet using the Wi-Fi network configured in `tools/dietpi/dietpi-wifi.txt` file (default: SSID=PhenoHive, Password=Phenohive).
- For a step-by-step guide on how to connect to the Raspberry Pi Zero W using SSH, see below.

Manual installation

1. Download the latest version of DietPi OS from the official website[12].
2. Flash the image on a microSD card using Balena Etcher[13].
3. Insert the microSD card in the Raspberry Pi and power it on.
4. Connect to the Raspberry Pi using a keyboard and a screen and follow the DietPi setup.

- A detailed guide on how to set up DietPi at the first boot is available on the official website[14].

[8] `https://dietpi.com/#downloadinfo`
[9] `https://www.balena.io/etcher/`
[10] `tools/dietpi/DietPi_files.md`
[11] `tools/dietpi/dietpi-wifi.txt`
[12] `https://dietpi.com/#downloadinfo`
[13] `https://www.balena.io/etcher/`
[14] `https://dietpi.com/docs/install/#3-prepare-the-first-boot`

F.5.2 Project setup

Automated setup

1. Ensure that the Raspberry Pi is connected to the internet; you can check the connection using `ping google.com`.
2. Clone the repository using `git clone https://github.com/Oldgram/PhenoHive.git PhenoHive`. If you don't have git installed, you can install it using `apt-get install git`.
3. Navigate to the PhenoHive folder using `cd PhenoHive`.
4. Run the setup script using `bash setup.sh`. This script will install the necessary packages, set up the system, and enable the service.

- *Note:* Due to the limited resources of the Raspberry Pi Zero W, the setup script can take up 2 to 3 hours to complete.
- If the setup was successful, you should see a blue message saying "Setup complete. A reboot is required before running the service.".
- If you encounter any error during the setup, follow the steps given in the manual setup section below.

5. Modify the configuration file `config.ini` according to your needs (see Configuration).
6. Run the calibration script[15] using `python3 tools/calibration.py` to calibrate the load cell (warning: this will overwrite the previous calibration coefficient).
7. Reboot the Raspberry Pi using `reboot`, the PhenoHive service will start automatically at boot.
8. The system is now running; you can check the status of the service using `systemctl status phenohive.service`.

Manual setup

1. Ensure that the Raspberry Pi is connected to the internet; you can check the connection using `ping google.com`.
2. Clone the repository using `git clone https://github.com/Oldgram/PhenoHive.git PhenoHive`. If you don't have git installed, you can install it using `apt-get install git`.
3. Navigate to the PhenoHive folder using `cd PhenoHive`.
4. Install the necessary packages:

```
sudo apt-get update
sudo apt-get install \
build-essential \
cmake \
gfortran \
pkg-config \
```

[15] `tools/calibration.py`

```
git \
python3 \
python3-pip \
python-is-python3 \
libopenblas-dev \
libatlas-base-dev \
patchelf \
ninja-build \
libcap-dev \
ffmpeg \
python3-dev \
python3-smbus \
python3-pil \
python3-rpi.gpio \
python3-av \
python3-libcamera \
python3-kms++ \
python3-pyqt5 \
python3-prctl \
python3-numpy \
python3-scipy \
python3-sklearn \
python3-skimage \
python3-statsmodels
```

5. Install the necessary Python packages:

```
sudo pip3 install --break-system-packages --no-cache-dir \
configparser==7.0.0 \
influxdb-client==1.44.0 \
hx711==1.1.2.3 \
pandas==1.5.1 \
adafruit-gpio==1.0.3 \
opencv-python==4.7.0.72 \
plantcv==3.14.3 \
mizani==0.9.0 \
plotnine==0.10.1
```

6. Download and install the ST7735 library:

```
git clone https://github.com/degzero/Python_ST7735.git
cd Python_ST7735
sudo python3 setup.py install
```

7. Enable the SPI interface and Picamera detection

- Depending on your system, it can be enables using the sytem configuration software such as `dietpi-config` (on DietPi) or `raspi-config` (on Raspberry Pi OS)

- You can also use the following commands to enable them manually:
```
echo "dtparam=spi=on" | sudo tee -a /boot/config.txt
echo "start_x=1" | sudo tee -a /boot/config.txt
echo "camera_auto_detect=1" | sudo tee -a /boot/config.txt
```

9. Create and enable PhenoHive service:

- Copy the phenohive.service[16] file to the systemd folder using `cp tools/phenohive.service /lib/systemd/system/`.
- Edit the service file using `nano /lib/systemd/system/phenohive.service` and replace:
  - `User=root` to your current user (on DietPi, you can ignore this step).
  - `PHENOHIVE_DIRECTORY` in the `WorkingDirectory` and `ExecStart` fields to the path of the PhenoHive folder.
  - The file should look like this (example for root user of DietPi):
```
[Unit]
Description=PhenoHive station service
Documentation=https://github.com/Oldgram/PhenoHive
After=multi-user.target
{
[Service]
User=root
WorkingDirectory=/root/PhenoHive/
ExecStart=/usr/bin/python /root/PhenoHive/main.py
Restart=on-failure

[Install]
WantedBy=multi-user.target
Alias=phenoHive.service
```
- Enable the service using:
```
sudo chmod 644 /lib/systemd/system/phenohive.service
chmod +x main.py
sudo systemctl daemon-reload
sudo systemctl enable phenohive.service
```

10. Modify the configuration file `config.ini` according to your needs (see Configuration).
11. Run the calibration script[17] using `python3 tools/calibration.py` to calibrate the load cell (warning: this will overwrite the previous calibration coefficient).
12. Reboot the Raspberry Pi using `reboot`, the PhenoHive service will start automatically at boot.

[16] `tools/phenohive.service`
[17] `tools/calibration.py`

13. The system is now running; you can check the status of the service using `systemctl status phenohive.service`.

F.5.3 SSH connection

1. Make sure that you are connected to the same network as the Raspberry Pi.
2. Find your current IP using `ifconfig /all` (on Windows or macOS) or `ip a` (on Linux).

- You should see an IP address in the form of `192.168.1.X`, `10.42.0.X` or similar. Example output of `ifconfig /all`:

```
PS C:\Users\goffi> ipconfig /all
[...]
Wireless LAN adapter Wi-Fi:


   Connection-specific DNS Suffix  . : home
   Description . . . . . . . . . . . : Killer(R) Wi-Fi 6 AX1650i 160MHz Wireless Ne
   Physical Address. . . . . . . . . : 70-A8-D3-1B-15-93
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2a02:a03f:c09a:cf01:6a01:a243:a90:2561(Prefe
   Temporary IPv6 Address. . . . . . : 2a02:a03f:c09a:cf01:9dad:a85e:25f2:bb1e(Pref
   Link-local IPv6 Address . . . . . : fe80::4101:1239:213e:be14%2(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.129.99(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Lease Obtained. . . . . . . . . . : 17 July 2024 09:40:54
   Lease Expires . . . . . . . . . . : 17 July 2024 10:40:52
   Default Gateway . . . . . . . . . : fe80::22b8:2bff:fe0b:9f62%2
                                       192.168.166.254
   DHCP Server . . . . . . . . . . . : 192.168.166.254
   DHCPv6 IAID . . . . . . . . . . . : 40937683
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2A-30-89-94-08-8F-C3-64-23-E1
   DNS Servers . . . . . . . . . . . : fe80::22b8:2bff:fe0b:9f62%2
                                       192.168.166.254
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

- If your IP is, for example `192.168.34.17`, than your IP range is `192.168.34.0/24`. In the above example, the IP is `192.168.166.254`, so the IP range is `192.168.166.0/24`.

3. Find the IP address of the Raspberry Pi:

- Using nmap[18], scan the network using `nmap -sn <YOUR_IP_RANGE>` (replace `<YOUR_IP_RANGE>` with your IP range found at step 2).
- The Raspberry Pi should appear in the list with "Raspberry Pi Foundation" next to the MAC Address, its IP will be just above.
- Example:

```
PS C:\Users\goffi> nmap -sn 192.168.166.0/24
Starting Nmap 7.95 ( https://nmap.org ) at 2024-07-17 09:47 Romance Summer Time
Nmap scan report for 192.168.166.16
Host is up (0.013s latency).
MAC Address: B8:27:EB:BF:6D:1A (Raspberry Pi Foundation)
Nmap scan report for 192.168.166.254
Host is up (0.017s latency).
MAC Address: 4E:A5:88:52:12:F3 (Unknown)
Nmap scan report for 192.168.166.174
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.92 seconds
```

  Here, the Raspberry Pi has the IP `192.168.166.16` (the first in the list).

4. Connect to the Raspberry Pi using SSH with `ssh root@<RASPBERRY_PI_IP>` (replace `<RASPBERRY_PI_IP>` with the IP found at step 3). Enter the password when prompted (default: phenohive).
5. You can close the connection at any time using `exit`.

[18] `https://nmap.org/`

Appendix G

DietPi configuration files

G.1 `Dietpi_files.md`

This document describes the different changes made to the DietPi configuration files to automate the setup.

G.2 Changes in `dietpi.txt`

G.2.1 Language/Regional options

Change keyboard layout to french

Line 14: `AUTO_SETUP_KEYBOARD_LAYOUT=gb` -> AUTO*SETUP*KEYBOARD_LAYOUT=fr‘

Change timezone to Brussels

Line 17: `AUTO_SETUP_TIMEZONE=Europe/London` -> AUTO*SETUP*TIMEZONE=Europe/Brussels‘

G.2.2 Network options

Disable Ethernet and enable wifi

Line 23: `AUTO_SETUP_NET_ETHERNET_ENABLED=1` -> `AUTO_SETUP_NET_ETHERNET_ENABLED=0`

Line 24: `AUTO_SETUP_NET_WIFI_ENABLED=0` -> `AUTO_SETUP_NET_WIFI_ENABLED=1`

Change wifi country code to Belgium

Line 28: `AUTO_SETUP_NET_WIFI_COUNTRY_CODE=GB` -> `AUTO_SETUP_NET_WIFI_COUNTRY_CODE=BE`

(optional) Change hostname

Line 41: `AUTO_SETUP_NET_HOSTNAME=DietPi -> AUTO_SETUP_NET_HOSTNAME=PhenoHive`

Disable delay at boot until network connection is established

This will speed up each boot, as the stations might not always have an internet connection.

Line 48: `AUTO_SETUP_BOOT_WAIT_FOR_NETWORK=1 -> AUTO_SETUP_BOOT_WAIT_FOR_NETWORK=0`

G.2.3 Software options

Change SSH server to OpenSSH (instead of Dropbear)

Line 80: `AUTO_SETUP_SSH_SERVER_INDEX=-1 -> AUTO_SETUP_SSH_SERVER_INDEX=-2`

G.2.4 Non-interactive first run setup

On first login, run update, initial setup and software installs without any user input

Line 117: `AUTO_SETUP_AUTOMATED=0 -> AUTO_SETUP_AUTOMATED=1`

Change global password to be applied for the system

line 125: `AUTO_SETUP_GLOBAL_PASSWORD=dietpi -> AUTO_SETUP_GLOBAL_PASSWORD=phenohive`

Software to automatically install (add each line after line 133)

```
AUTO_SETUP_INSTALL_SOFTWARE_ID=17   # Git
AUTO_SETUP_INSTALL_SOFTWARE_ID=69   # RPi.GPIO
AUTO_SETUP_INSTALL_SOFTWARE_ID=74   # InfluxDB
AUTO_SETUP_INSTALL_SOFTWARE_ID=77   # Grafana
AUTO_SETUP_INSTALL_SOFTWARE_ID=130  # Python 3 pip
```

G.2.5 Misc DietPi program settings

(optional) Opt-out of DietPi-Survey

Line 140: `SURVEY_OPTED_IN=-1 -> SURVEY_OPTED_IN=0`

G.2.6 DietPi-Config settings

(optional) Disable Daily check for DietPi updates

Line 198: `CONFIG_CHECK_DIETPI_UPDATES=1 -> CONFIG_CHECK_DIETPI_UPDATES=0`

(optional) Disable Daily check for APT package updates

Line 202: `CONFIG_CHECK_APT_UPDATES=1` -> `CONFIG_CHECK_APT_UPDATES=0`

G.3 Changes in `dietpi-wifi.txt`

G.3.1 Configure Wifi connection

This requires wifi to be enabled in `dietpi.txt` (see above[1])

For each entry

One entry = one wifi, here the example is for entry 0. You do not need to change each field.

SSID: `aWIFI_SSID[0]='PhenoHive'`

Key (password): `aWIFI_SSID[0]='PhenoHive'`

G.4 Changes in `config.txt`

G.4.1 Enable picamera module and picamera detection

Warning: be sure to remove the leading '#'

Line 59: `#start_x=1` -> `start_x=1`

Add line 60: `camera_auto_detect=1`

G.4.2 Enable SPI interface

Warning: be sure to remove the leading '#'

Line 78: `#dtparam=spi=off` -> `dtparam=spi=on`

[1] `#disable-ethernet-and-enable-wifi`

Bibliography

[1] Martin Lallemand and Louis Lemaire. "Innovation pédagogique dans le cadre des travaux pratiques de physiologie végétale : développement d'une station de phénotypage pour plante individuelle". Français. MA thesis. UCL - Faculté des bioingénieurs, 2023. URL: http://hdl.handle.net/2078.1/thesis:40992.

[2] *What is Phenotyping? – The Department of Molecular & Comparative Pathobiology*. en-US. URL: https://mcp.bs.jhmi.edu/what-is-phenotyping/ (visited on 08/06/2024).

[3] Lei Li, Qin Zhang, and Danfeng Huang. "A Review of Imaging Techniques for Plant Phenotyping". en. In: *Sensors* 14.11 (Oct. 2014), pp. 20078–20111. ISSN: 1424-8220. DOI: 10.3390/s141120078. URL: https://www.mdpi.com/1424-8220/14/11/20078 (visited on 08/03/2024).

[4] *Course: LBIR1251 = Physiologie végétale*. URL: https://moodle.uclouvain.be/course/view.php?id=3384& (visited on 08/05/2024).

[5] *Plant Phenotyping | Photon Systems Instruments*. URL: https://plantphenotyping.com/ (visited on 07/15/2024).

[6] *Phenospex - Smart plant analysis & Phenotyping systems*. URL: https://phenospex.com/ (visited on 07/15/2024).

[7] *WIWAM – Automated systems for plant phenotyping – WIWAM presents phenotyping platforms for plant phenotype analysis*. URL: https://www.wiwam.be/ (visited on 07/16/2024).

[8] *Plant phenotyping facility*. URL: https://uclouvain.be/en/research-institutes/eli/elia/rootphair.html (visited on 07/14/2024).

[9] Laboratory for Plant Ecophysiology under Environmental Stress (LEPSE). *Montpellier Plant Phenotyping Platforms (M3P)*. URL: https://eng-lepse.montpellier.hub.inrae.fr/platforms-m3p/montpellier-plant-phenotyping-platforms-m3p (visited on 07/14/2024).

[10] UMR Agroecologie. *Serre-4PMI - UMR Agroecologie*. URL: https://umr-agroecologie.dijon.hub.inrae.fr/plateformes-et-infrastructures/plateformes/serre-4pmi (visited on 07/14/2024).

[11] Wageningen Univeristy and Reasearch. *The Netherlands Plant Eco-phenotyping Centre: NPEC*. June 2019. URL: https://www.wur.nl/en/product/the-netherlands-plant-eco-phenotyping-centre-npec.htm (visited on 07/14/2024).

[12] Donald Danforth Plant Science Center. *About*. en-US. URL: https://plantcv.org/about (visited on 08/06/2024).

[13] *Home*. en-US. URL: https://opencv.org/ (visited on 08/06/2024).

[14] Malia A. Gehan et al. "PlantCV v2: Image analysis software for high-throughput plant phenotyping". en. In: *PeerJ* 5 (Dec. 2017), e4088. ISSN: 2167-8359. DOI: 10.7717/peerj.4088. URL: https://peerj.com/articles/4088 (visited on 08/06/2024).

[15] Chong Teng, Noah Fahlgren, and Blake C. Meyers. *Tasselyzer, a machine learning method to quantify anther extrusion in maize, based on PlantCV*. en. Sept. 2021. DOI: 10.1101/2021.09.27.461799. URL: http://biorxiv.org/lookup/doi/10.1101/2021.09.27.461799 (visited on 08/06/2024).

[16] Min-guo Liu et al. "Analyzing architectural diversity in maize plants using the skeleton-image-based method". en. In: *Journal of Integrative Agriculture* 22.12 (Dec. 2023), pp. 3804–3809. ISSN: 20953119. DOI: 10.1016/j.jia.2023.05.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S2095311923001429 (visited on 08/06/2024).

[17] *InfluxDB | Real-time insights at any scale*. Jan. 2022. URL: https://www.influxdata.com/home/ (visited on 08/06/2024).

[18] *InfluxDB Time Series Platform*. Jan. 2023. URL: https://www.influxdata.com/products/influxdb/ (visited on 08/06/2024).

[19] *Use the InfluxDB Python client library | InfluxDB Cloud (TSM) Documentation*. URL: https://docs.influxdata.com/influxdb/cloud/api-guide/client-libraries/python/ (visited on 08/06/2024).

[20] *Grafana: The open observability platform*. en. URL: https://grafana.com/ (visited on 08/06/2024).

[21] *Get started with Grafana and InfluxDB | Grafana documentation*. en. URL: https://grafana.com/docs/grafana/latest/getting-started/get-started-grafana-influxdb/ (visited on 08/06/2024).

[22] UCL. *Plant physiology*. EN. text. Aug. 2024. URL: https://uclouvain.be/en-cours-2024-lbir1251 (visited on 08/02/2024).

[23] Ikhlas Ghiat, Hamish R. Mackey, and Tareq Al-Ansari. "A Review of Evapotranspiration Measurement Models, Techniques and Methods for Open and Closed Agricultural Field Applications". en. In: *Water* 13.18 (Sept. 2021), p. 2523. ISSN: 2073-4441. DOI: 10.3390/w13182523. URL: https://www.mdpi.com/2073-4441/13/18/2523 (visited on 08/06/2024).

[24] DegrawSt. *Arduino Scale With 5kg Load Cell and HX711 Amplifier*. en. URL: https://www.instructables.com/Arduino-Scale-With-5kg-Load-Cell-and-HX711-Amplifi/ (visited on 08/06/2024).

[25] yida. *10 Things you can do with your HX711 and Load Cell*. en-US. Nov. 2019. URL: https://www.seeedstudio.com/blog/2019/11/26/10-things-you-can-do-with-your-hx711-and-load-cell/ (visited on 08/06/2024).

[26] *TAL226 Miniature parallel beam load cell*. URL: http://htc-sensor.com/products/142.html (visited on 08/06/2024).

[27] alldatasheet.com. *HX711 Datasheet(PDF)*. en. URL: http://www.alldatasheet.com/datasheet-pdf/pdf/1132222/AVIA/HX711.html (visited on 08/06/2024).

[28] Achim Walter, Frank Liebisch, and Andreas Hund. "Plant phenotyping: from bean weighing to image analysis". In: *Plant Methods* 11.1 (Mar. 2015), p. 14. ISSN: 1746-4811. DOI: 10.1186/s13007-015-0056-8. URL: https://doi.org/10.1186/s13007-015-0056-8 (visited on 08/03/2024).

[29] *The Picamera2 Library - github.com*. URL: https://github.com/raspberrypi/picamera2 (visited on 08/03/2024).

[30] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[31] Raspberry Pi Foundation. *Raspberry Pi OS - Official Website*. URL: https://www.raspberrypi.com/software/ (visited on 07/28/2024).

[32] DietPi Project Team. *DietPi - Official Website*. URL: https://dietpi.com/ (visited on 07/29/2024).

[33] DietPi Project Team. *Alpine Linux - Official Website*. URL: https://alpinelinux.org/about/ (visited on 07/29/2024).

[34] Bela Markus. *piCore - README - tiny core linux*. URL: http://tinycorelinux.net/5.x/armv6/releases/README (visited on 07/30/2024).

[35] *« Linux Kernel » - Endoflife.Date*. URL: https://endoflife.date/linux (visited on 07/28/2024).

[36] Bill Dyer. *What are Daemons in Linux? Why are They Used?* Sept. 2023. URL: https://itsfoss.com/linux-daemons/ (visited on 08/01/2024).

[37] Software Freedom Conservancy. *Git - gitignore Documentation*. Sept. 2020. URL: https://git-scm.com/docs/gitignore (visited on 07/27/2024).

[38] Cameron McKenzie. *Use .gitkeep to commit and push an empty Git folder or directory*. Sept. 2020. URL: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/gitkeep-push-empty-folders-git-commit.

[39] Pandas Community. *Pandas docstring guide; pandas 2.2.2 documentation — pandas.pydata.org*. URL: https://pandas.pydata.org/docs/development/contributing_docstring.html (visited on 07/26/2024).

[40] Josh Di Mella. *A Guide to Python Docstring Formats: Choosing the Right Style for Your Code — joshdimella.com*. 2023. URL: https://joshdimella.com/blog/python-docstring-formats-best-practices (visited on 07/26/2024).

[41] Georg Brandl. *GitHub - sphinx-doc/sphinx: The Sphinx documentation generator — github.com*. URL: https://github.com/sphinx-doc/sphinx (visited on 07/26/2024).

[42] David Goodger. *PEP 287 – reStructuredText Docstring Format | peps.python.org — peps.python.org*. 2002. URL: https://peps.python.org/pep-0287/ (visited on 07/26/2024).

[43] Python Software Foundation. *typing — Support for type hints — docs.python.org*. URL: https://docs.python.org/3/library/typing.html (visited on 07/27/2024).

[44] C. Giridhar. *Learning Python Design Patterns*. Community experience distilled. Packt Publishing, 2016. ISBN: 9781785887376. URL: https://books.google.be/books?id=161KDAAAQBAJ.

[45] Python Software Foundation. *configparser — Configuration file parser*. URL: https://docs.python.org/3/library/configparser.html (visited on 07/27/2024).

[46] *Raspberry Pi Zero 2 with HEADER- Wireless + Cam conn*. en. URL: https://shop.mchobby.be/en/motherboards/2334-raspberry-pi-zero-2-with-header-wireless-cam-conn-3232100023345.html (visited on 08/13/2024).

[47] *balenaEtcher - Flash OS Images to SD Cards & USB Drives*. URL: https://www.balena.io/etcher (visited on 07/30/2024).

[48] *Find Raspberry Pi computers in stock*. en. URL: https://rpilocator.com/?country=BE&cat=PIZERO (visited on 08/07/2024).

[49] Les Pounder last updated. *How to Turn a Raspberry Pi Into a Wi-Fi Access Point*. en. Sept. 2022. URL: https://www.tomshardware.com/how-to/raspberry-pi-access-point (visited on 08/09/2024).

[50] *Install InfluxDB 2 on Raspberry Pi | Random Nerd Tutorials*. en-US. June 2022. URL: https://randomnerdtutorials.com/install-influxdb-2-raspberry-pi/ (visited on 08/09/2024).

[51] Sandy Macdonald. *Setting up InfluxDB and Grafana on the Raspberry Pi 4*. URL: http://sandyjmacdonald.github.io/2021/12/29/setting-up-influxdb-and-grafana-on-the-raspberry-pi-4/ (visited on 08/09/2024).

[52] *Flux Documentation*. URL: https://docs.influxdata.com/flux/v0/ (visited on 08/13/2024).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**