UCLouvain

École polytechnique de Louvain

La Plante Connectée

An intuitive low-cost phenotyping station

Author: **Louis Colinet**
Supervisor: **Cristel Pelsser**
Readers: **Xavier Draye, Laurent Francis**
Academic year 2024–2025
Master [120] in Electrical Engineering

Abstract

As part of UCLouvain's bioengineering program, hands-on learning about plant physiological phenomena is essential but remains limited by material, logistical, and financial constraints. This thesis is a continuation of the PhenoHive project [1, 2], a compact, intuitive, and low-cost plant phenotyping station, initially developed to provide each group of students with a personal plant observation tool.

This work presents the improvement of the existing station by addressing both hardware and software aspects. The main contributions include the integration of new sensors (humidity, light), the optimization of image capture and growth analysis via PlantCV, online export of the database already implemented with InfluxDB and Grafana in the previous works [1, 2], and the improvement of the user-friendliness of the interface and the physical design.

The results show that it is possible to provide students with a robust, scalable, and cost-effective tool that promotes autonomy and engagement in learning plant phenotyping.

Acknowledgements

I would first like to thank Cristel Pelsser, my supervisor, for her support throughout this project. Her wise advice, proofreading, and regular monitoring were essential to the completion of this thesis.

I also thank Xavier Draye, the thesis reviewer, for his insightful comments and for the clarifications provided regarding the project's objectives.

Many thanks to Laurent Francis for reading the thesis and providing his feedback.

Thank you to the ELEC club for their help in soldering the components, and to Makilab, particularly Jérôme Leclère, for 3D printing the various parts of the station.

I would also like to thank Marc Migon for providing a corn plant, as well as Nicolas Biot for providing the prototypes and his valuable insights into the project.

Finally, I would like to thank Dylan Goffinet for his explanations at the start of the project, which allowed me to better understand the challenges and technical foundations.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A bioengineer needs to possess a thorough understanding of plant life. This understanding includes the biological mechanisms that control their growth. Consequently, practical training on a concept called phenotyping is incorporated into their studies. In these training sessions, learners have the opportunity to witness these biological mechanisms firsthand, through direct observation or with the aid of sensors. Nevertheless, current methods tend to be expensive and not easily portable, which limits the effectiveness of this hands-on learning experience.

In this context, Louis Lemaire and Martin Lallemand came up with a brilliant concept for their thesis, which involved creating a compact and affordable mini-phenotyping station. Their model would enable a set of students to examine a plant with their personal station [1].

At the start, they decided to implement two devices: a load cell scale and a small camera. The scale tracks the evapotranspiration emitted by the plant during its respiration, and the small camera, utilizing image recognition technology, assesses the plant's development. A Raspberry Pi was selected to serve as the main processing hub to link all these components.

The following year, in 2023-2024, the project was taken over by Dylan Goffinet. During his year of work, he focused primarily on the software aspect. Indeed, the main improvements he made were a change of OS to a lighter version, better adapted to the limited resources of the Raspberry Pi used, a simplification and automation of the station's computer setup, and a clarification of the code received, which was initially quite messy. For the latter, he divided the code into several sub-files and added comments to most functions. These improvements greatly facilitated code comprehension and debugging [2].

In this overview, we will start by explaining the objectives of the project related to the bioengineering course as well as a presentation of some existing tools with a similar purpose. Then, we will give a more detailed explanation of the current setup, solve persistent

problems, improve the quality of image capture through different approaches, introduce two additional sensors, one for humidity and one for light, which will allow us to collect more data, we will also allow the station to be able to send data to a remote server and, finally, we will create a new, more compact and robust station enclosure.

Finally, to follow the changes made to the code, which will be mentioned in the rest of the report, we share the following link which leads to the GitHub hosting the project.

Current GitHub repository: `https://github.com/locolinet/PhenoHive.git`

Chapter 2

State of art

In this section, we will place the PhenoHive station project in its initial context. We will explain the need of bioengineering students for such a station and define the concept of phenotyping. Finally, we will present the objectives set at the launch of the project.

2.1 Need in the bioengineering curriculum

During the second year of their first cycle of studies entitled "Bachelier en sciences de l'ingénieur, orientation bioingénieur" students are required to follow the course LBIR1251 called "Biologie et Physiologie végétale" This learning unit is taught in the second term and counts for 5 credits [10, 11]. The teachers are Xavier Draye, Stanley Lutts, Lola Leveau and Nicolas Biot. The latter two are more specifically responsible for practical work.

The course content covers different aspects [11]:

- Plant water relations, water potential and its components, water transport from the soil to the atmosphere via the plant, and stomatal regulation. The importance of these processes at the cellular and tissue levels is also discussed.
- Mineral nutrition is detailed, highlighting the interaction between roots and soil, the role of essential elements, and the mechanisms of nutrient transport through cells.
- Photosynthesis is described in two phases: the light phase, linked to the structure of photosynthetic organs, and the dark phase, which takes into account gas exchange and water use efficiency.
- Assimilate transport is presented, explaining how they are loaded and unloaded in the phloem, and how their distribution is regulated according to the plant's needs.

2.1.1 Practical part

Practical work is given both in the greenhouse for handling plants and measuring tools in order to collect data, but also in the auditorium for analyzing them. To follow these sessions, students are divided into groups of three which will remain the same until the end of the term. These sessions are organized as shown in Fig. 2.1:

Semaines	Thématiques	Séances
(S2 ou S3)* et S4	L'eau	- Une séance en serres (manipulation) - Une séance en auditoire (analyse)
(S5 ou S6)*	La nutrition minérale	- Une séance en serres (manipulation)
(S7 ou S8)* et S10	La photosynthèse	- Une séance en serres (manipulation) - Une séance en auditoire (analyse)
S9 et S11	Créer un protocole d'expérience	- Deux séances en auditoire
S12	L'environnement	- Une expérience à réaliser à domicile durant tout le quadrimestre - Une séance en auditoire (analyse)

Figure 2.1: Organization of practical work divided according to the five thematic blocks [3]

We can see that the practical work is divided into five thematic blocks: water, mineral nutrition, photosynthesis, creation of an experimental protocol and finally the environment.

Water

For the first topic, water, students must follow the development of a corn plant by measuring leaf growth, leaf transpiration, and leaf stomatal conductance, and observe these variables as a function of time of day, air humidity, and air temperature [3].

To collect all this data, students use different sensors [3]. One of these is the porometer. In our case, the porometer is used to measure relative humidity and temperature in a small, airtight chamber placed around a leaf. If the measured humidity increases quickly, this means that vapor diffusion from the plant is high.

The second is a potentiometer. This is used to measure the height of the plant. To do this, students attach a clip to the tallest leaf of the plant, then connect this clip to a wire that passes over a pulley and is attached at the other end to a small weight. As the plant grows, it will slightly rotate the pulley, which, itself connected to the potentiometer, will vary its internal resistance, which can be measured and, once measured, can give us the relative growth of the plant.

The third are scales and dataloggers. These are used to measure evapotranspiration. Evapotranspiration is defined as: "The combined processes through which water is transferred to the atmosphere from open water and ice surfaces, bare soil and vegetation that make up the Earth's surface."[12]. In our case we have no evaporation due to free water and ice surfaces. A weight measurement is recorded every 20 minutes and allows us to see how much water is lost. However, it's important to carefully note watering, which increases the weight.

The last sensors are more general humidity and temperature sensors used for greenhouses. Data is recorded every 10 minutes.

2.1.2 Mineral nutrition and mineral assimilation

For this second theme, students observe the behavior of different plants such as tomato, pea, and corn plants when faced with certain nutrient intakes or even deficiencies. To do this, they study, for example, the roots or the color of the leaves. One of the experiments also uses a semi-hydroponic circulatory culture.

2.1.3 Photosynthesis

The main objective of this thematic block is for students to become familiar with certain measuring tools.

The first of these is the chlorophyll meter. This measuring device is obviously used to measure the chlorophyll content in a leaf and is based on a simple operation. On one side of the leaf, two lights are emitted: one red, which is absorbed by the chlorophyll, and the other infrared, but not absorbed by the chlorophyll. On the other side, a sensor detects the portion of red light transmitted through the leaf, allowing the chlorophyll content to be calculated. The infrared light, on the other hand, allows for a calibration based on the leaf's thickness.

The second sensor is a porometer, the operation of which has already been explained above.

The third sensor is the IRGA, an acronym for InfraRed Gas Analyzer. Like the porometer, this sensor is designed as a clamp that creates a sealed chamber around the leaf and is used to measure the vapor and CO2 released by the leaf. To measure the quantity of these gases, the IRGA uses infrared light, which will be absorbed to a greater or lesser extent depending on their content. This measurement is performed on an airflow at both the inlet and outlet, allowing the difference to be calculated.

The last sensor is a florimeter, which measures the portion of light energy absorbed by a plant's leaf. To do this, it first sends a beam of light and then calculates the portion of light reflected and transmitted through the leaf. The "lost" portion is therefore the absorbed part.

2.1.4 Environment

For this topic, students study the behavior of a sunflower in different environments. Each student receives a plant and grows it in the room of their choice. This creates a wide variety of environments. During the experiment, which lasts approximately six weeks, the students must monitor the plant's temperature, watering, and growth.

2.2 Phenotyping

To fully understand the purpose of the PhenoHive station, it is important to understand what phenotyping is. Phenotyping is a method used to observe and measure the physical and biological characteristics of an organism, such as a plant or an animal. These characteristics are actually called phenotypes and are defined by: "The phenotype describes all the observable traits of an individual. It depends on gene expression (genotype) and the environment."[13]. Phenotypes therefore depend not only on the plant's genetic makeup but also on environmental conditions.

For our research station project, we can focus more simply on plant research. In our case, phenotyping is used to study aspects such as plant growth, shape, color, disease resistance, and their ability to absorb water and nutrients. Phenotyping is therefore essential for improving crops, selecting new varieties, and optimizing cultivation methods.

Thanks to technological advances, phenotyping now uses tools such as specialized cameras, sensors, and analysis software. These technologies enable the collection of accurate, real-time data, helping researchers and farmers better understand plants and improve their production.

Many industrial companies around the world are already actively addressing this issue. For your information, the IPPN (International Plant Phenotyping Network) website [4] provides an interactive map listing numerous research groups involved in plant phenotyping, in one way or another. This map is shown in Fig 2.2.

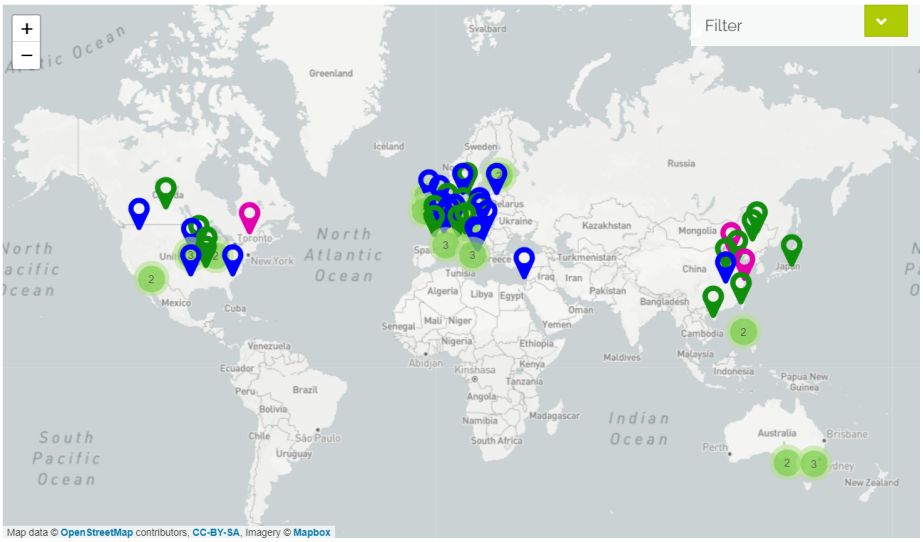


Figure 2.2: Map from the IPPN website showing research groups dedicated to plant phenotyping [4]

To begin with, in Louvain-la-Neuve, we have the ELIA group, Earth and Life Institute which is affiliated with UCLouvain [14]. This one performs a phenotyping of a plant's roots. To do this, the technology called "RootPhAir" is used. This plants grows in aeroponics. Aeroponics is: "the process of cultivating plants in an air or mist environment, eliminating the need for soil or an aggregate medium." [15]. Next, each of the many plants is brought

in front of the camera, which takes a photo of the roots. This process is repeated every two hours for each plant. In addition, this technology uses backlighting, meaning lighting is positioned behind the plant relative to the camera.

Another example, still in Belgium, is located in Gent and is therefore affiliated with the UGent University. The system used here is WIWAM maize [16].This system uses automated imaging, meaning that a robotic arm picks up each plant individually and brings it to a dedicated imaging area. Once the plant is in this area, a multi-view RGB (Red, Green, Blue) image is taken. This image capture allows for the 3D reconstruction of the plant. The robotic arm then places the plant back in its original location. The capacity of a table containing plant pots can be up to 156 pots. The imaging area is not the only area, the robotic arm also takes each plant to another area where the plant will be weighed and irrigated. Thanks to this system, the plants can then have different soil moisture levels according to predefined choices.

WIWAM maize is one of the UGent systems, but we can also cite the Arabidopsis WIWAM [17] or the Phenovision [18]

As we can see in Fig. 2.2 it is not only in Belgium that we find phenotyping systems. For example, in Denmark, we also find a phenotyping system at the University of Copenhagen [19]. This system also uses, like the two systems presented above, an individual movement of each plant towards a measurement zone. This area is a closed imaging box equipped with thermal and multispectral cameras to assess temperature, fluorescence, and various physiological parameters. A system of LEDs and fluorescence filters allows for various tests. Gas exchange sensors can also be used. The system can manage up to 117 plants simultaneously, weighing, watering, and measuring them up to twice a day. Watering is automatically adjusted based on soil moisture measured by conductivity.

There are many other systems and groups working on this topic. While the three systems presented here are affiliated with European universities, these groups are also found in America and Asia, and may also be private or not.

2.3 Zea maize

It is explained in the two previous theses [1, 2], why the maize plant was chosen for the project. The chosen variety is more precisely the *zea maize* which has the particularity of being robust but above all of growing quickly and more or less on a plane, that is to say that the leaves always grow on one side and then on the opposite side which forms the plane in question. This plane criterion will be very useful for the growth parameter that we will use for phenotyping.

2.4 Launch of the project

While several systems exist and several experiments can already be conducted in greenhouses, why was it necessary to develop a system like PhenoHive?

According to the initiators of this project, Martin Lallemand and Louis Lemaire, in their final thesis, various problems and expectations are highlighted[1].

First, greenhouses are a limiting factor. According to the course assistants, they do not have enough light to demonstrate certain phenomena. They are also not very spacious and cannot accommodate all the students enrolled in the course at once. This is why the practical sessions must sometimes be spread over two weeks, as we can see for the first three themes in the Fig. 2.1.

A second aspect is that the sensors used in greenhouses, such as the porometer, IRGA, fluorimeter, etc., can be very expensive and sometimes fragile. The goal of the PhenoHive station is therefore to be able to mimic certain experiments conducted in greenhouses but using less expensive and more robust sensors.

A third point is that students really appreciate the interactivity and the experiments carried out in the greenhouses. They also find that these are sometimes short and that they do not always participate as much as they would like.

It can also be noted that at present, the data collected during the experiments are recorded manually in tables such as Excel or other, which is not very practical and a source of errors.

Ultimately, all these points justify the creation and design of a robust and inexpensive prototype station that could be shared among 3-4 students. They could then have their own individual corn plants and could carry out their experiments throughout the semester.

Chapter 3

Initial prototype

In this section, we will introduce you to the station, the prototype as initially received. First, we will present the hardware, including the various components inside and outside the box. In the second section, we will discuss the software side. We will talk about both the code itself and the interface directly offered to the user. A section on data storage will also be discussed.

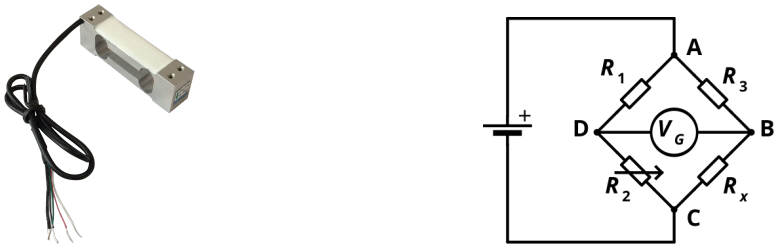
3.1 Hardware

From the outside, the station is mainly composed of two parts, the scale, which is the yellow object in Fig. 3.1, and the station itself, which is gray and has two buttons and a screen.

Figure 3.1: PhenoHive Station exterior view

3.1.1 The scale

The heart of it is a TAL226 load cell[1]. To accurately detect small resistance variations, the load cell's strain gauges are arranged in a Wheatstone bridge configuration, Fig.3.2b. In a no-load state, the bridge is balanced and the output voltage is negligible or zero. When a force is applied, the strain gauges deform, altering their resistance and unbalancing the bridge. This generates a small voltage difference proportional to the applied force. The TAL226 model was selected over other options following experiments conducted in previous research on this topic [1].



(a) (b)

Figure 3.2: (a) TAL226 load cell [5], (b) Wheatstone Bridge circuit [6]

The load cell produces a very low analog signal, so we use the HX711 module to amplify and convert it into digital data for the microcontroller. Specifically designed for weighing applications, the HX711 ensures high accuracy in weight measurements [21].

In the first prototype, Louis Lemaire and Martin Lallemand proposed a specially designed box to accommodate the TAL226 load cell. For this, they created a 3D model that could be printed. It is designed to compress by a maximum load of 3 kg. The model is shown in Fig. 3.3.

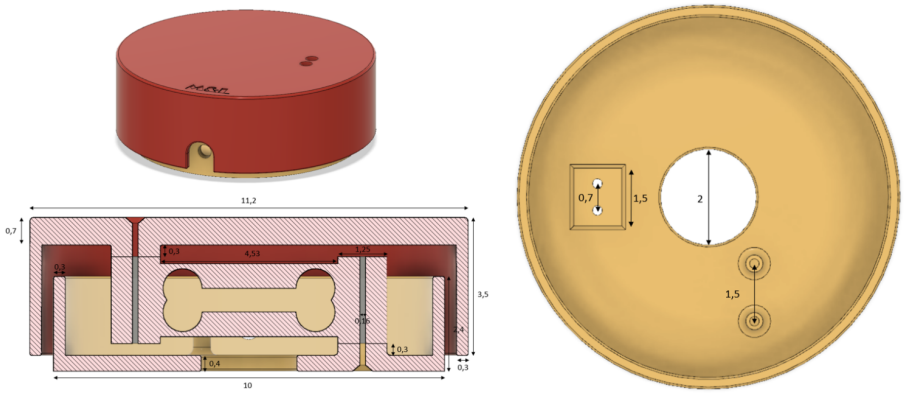


Figure 3.3: 3D model of the scale including dimensions [1]

[1] "A load cell is a sensor used to measure force or weight. Its operation relies on strain gauges, which detect mechanical deformations of an object by changing their electrical resistance in response to stretching or compression. This change in resistance is then converted into an electrical signal proportional to the applied force" [20].

3.1.2 The station

The station itself is composed of five main elements. The interior of the box is shown in Fig. 3.4. The elements are the buttons, the camera, the screen, the Raspberry Pi and the relay.

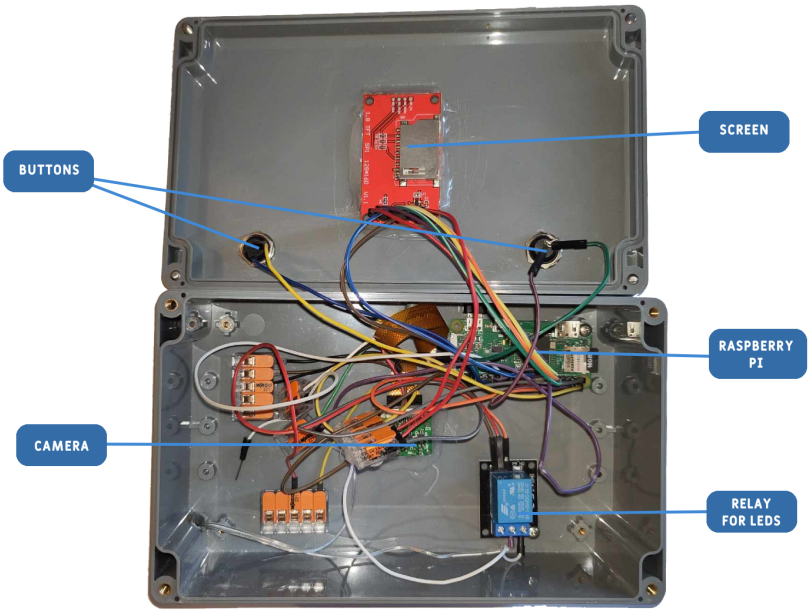


Figure 3.4: Interior of the station showing the different elements

First, we have the Raspberry Pi. This acts as a central unit. This means that it receives and transmits signals picked up by sensors and various external elements. It is also the one that will contain the code as well as the data. To store this data in memory, we must insert an SD card. It is also thanks to this that we will be able to initialize the station (explained in [2]). For this project, the Raspberry Pi Zero W model was chosen. This choice was made on the basis of its low price and that the needs such as Wi-Fi or compatibility with sensors as well as Python were met. We will see, however, that this being restrictive for certain improvements, we had to change it.

There are two buttons, each enabling a different choice in the menus, which will be detailed later. On the same side, the screen displays these menus.

On the opposite side, we have the camera. This is used to capture images of the plant. These images are sent to the Raspberry Pi, which will analyze them to determine the plant's size. The camera chosen is a Pi camera. A relay controls an LED strip around the camera to provide constant illumination, since the Pi's 3.3 V GPIO cannot directly power the 5 V strip.

A wiring diagram is available in the appendix A. Please note, however, that some changes have been made, and this therefore corresponds to the current circuit diagram. To find the old wiring diagram, you can refer to the appendices in [2].

All of these elements are simply contained in a box used by electricians. To pass through the buttons, screen, camera, power supply, and wire to the scale, the openings in the enclosure were manually drilled to accommodate the components.

Finally, to clearly demonstrate that the objective of a "low-cost phenotyping station" is achieved for the moment, we have taken the table from the previous thesis [2] in which we updated the prices (March 26, 2025). You will find this table in the appendix C. The total cost comes to 57.13€, which is well-suited for a low-cost station. In comparison, we can find existing solutions ranging from a few hundred to several thousand dollars[22, 23].

3.2 Software

In the software section, we will first show you how to digitally initialize the station, then we will detail each file belonging to the project and finally we will finish with a presentation of the interface and its different menus.

3.2.1 Initialization

During his work, Dylan Goffinet developed several ways to initialize the station. In this section, we will show you only the simplest method. If you wish, you can also read the other, more manual methods explained by Dylan Goffinet in his thesis [2].

- **Step 1:** The first step of the installation is to download the SD card image[2] containing the project and the Raspberry Pi configuration files.

  Image to download: `https://uclouvain-my.sharepoint.com/:u:/g/personal/dylan_goffinet_student_uclouvain_be/Eb_mK8L4GptGhppHbfrdEPoBmDWEppfjG-rZpERCGurvsw?e=Wdi3cV`

- **Step 2:** The second step is to use Balena Etcher[3] to copy the image downloaded in step 1 to the SD card.

- **Step 3:** Step 3 is simply a matter of inserting the freshly flashed SD card into the slot on the Raspberry Pi. You can then plug in the Raspberry Pi, which should automatically connect to the PhenoHive Wi-Fi network, whose password is also PhenoHive. This Wi-Fi network is generated by another Raspberry Pi.

- **Step 4:** Finally, for this last step, you need to connect via ssh to the workstation which is explain in [2]. Once this is done, you will need to navigate to the folder containing all the project files. To do this, use the command:

  cd PhenoHive

[2] A memory image is actually a bit-for-bit copy of an existing memory on another receiving memory.

[3] Balena Etcher, which is open-source software, is an image writing tool. This tool is actually a software program designed to write disk image files, such as ISOs or IMGs, to removable storage media, such as USB drives and SD cards. It's also particularly intuitive thanks to its simple interface. To download it, use the following link: `https://etcher.balena.io/#download-etcher`

Then, since the project is hosted on GitHub[4] and the folder is therefore a repository, you can use the command:

git pull

Once the order is completed, the project should now be up to date with the latest version. To take into account the changes, it is necessary to restart the station. Use the command:

reboot

3.2.2 Grafana and InfluxDB

To collect data, the station will send all its measurements over Wi-Fi. They will then be saved to InfluxDB and can be easily viewed on Grafana. A diagram is shown in Fig. 3.5

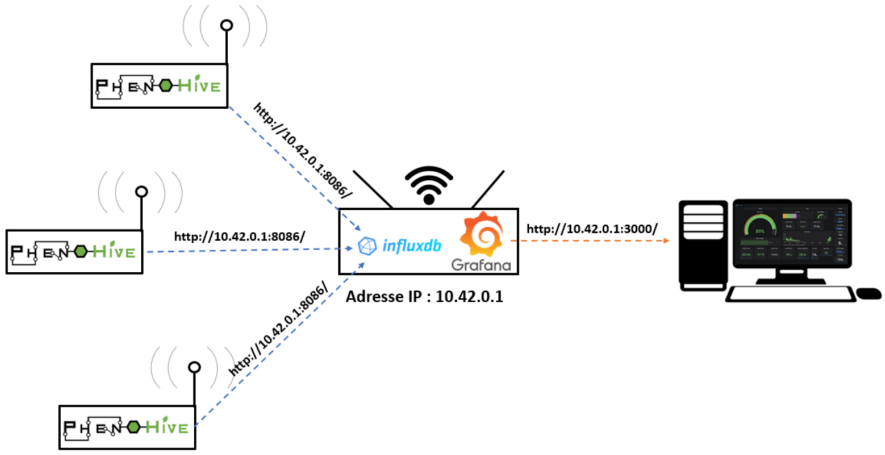


Figure 3.5: Connection between stations, central unit and user [1]

InfluxDB

InfluxDB is a time series database developed by InfluxData. It is particularly suited for use cases such as the Internet of Things (IoT), application measurement, and financial or scientific data analysis. InfluxDB is based on a storage engine that enables advanced compression and fast queries using a specific language called InfluxQL, similar to SQL, as well as Flux, a more flexible language for data processing. One of InfluxDB's major strengths is its easy integration with monitoring tools like Grafana or Telegraf, facilitating the visualization and exploitation of metrics.[24, 25].

InfluxDB is therefore particularly well-suited to the PhenoHive project due to its ability to efficiently manage time series, which are central to monitoring plant and environmental parameters. In PhenoHive, the Raspberry Pi captures images and collects associated data such as growth, weight, and other variables relevant to plant phenotypic analysis. All this data is time-stamped and requires optimized storage for subsequent access and analysis.

[4]GitHub is an online platform that allows developers to store, share, and collaborate on code using Git, a version control tool. It facilitates teamwork with features like change tracking, pull requests, and project management.

Grafana

Grafana is an open-source data visualization and analysis tool primarily designed to monitor metrics from a variety of sources. It allows users to create interactive and dynamic dashboards by aggregating data from databases. Widely used in IT infrastructure monitoring and observability, Grafana facilitates real-time analysis thanks to its advanced query capabilities and numerous customizable chart types. Being flexible and extensible, it integrates easily with other monitoring tools and offers sharing and collaboration options for efficient data management.

As part of the PhenoHive project, Grafana enables real-time visualisation of captured data. This provides students with an intuitive interface for monitoring the status of the system and adapting their interventions based on their observations.

Actual use

Currently, the operation of the connections between the central unit, the stations and the user works like this:

- All three units must be connected to the same wifi network, i.e. the *PhenoHive* network as mentioned above.
- On the station side, they will have in a config.ini file the URL corresponding to InfluxDB as well as the token corresponding to the database and allowing access to it. In this file, there is also a variable called *id* which allows you to know which station is sending which data. This file will be explained in more detail in the next section. They will therefore be able to communicate with the central unit which is in this case a Raspberry Pi 3.
- The central unit here, in addition to being the hotspot (*PhenoHive* network), is the unit that manages InfluxDB and Grafana. It therefore receives the measurements made by the stations and distributes them according to their ID. Then we retrieve the data stored on influxDB to Grafana. On Grafana we have the possibility to create dashboards per station that students can access in a controlled manner. An example of a dashboard is shown in Fig. 3.6

3.2.3 Role of each file

For a better view and understanding, the Phenohive project is divided into several files and directories. Here we will only explain the files that will be significantly modified during this thesis. A more detailed explanation of each file is available in the appendix B. All these files are available at the following GitHub link but we will still explain them one by one:

PhenoHive Git: `https://github.com/Oldgram/PhenoHive/tree/main`

Figure 3.6: Example of a possible dashboard on Grafana

To more easily follow the file explanations, a tree structure is provided at the end of the section.

- **setup.config**: This file contains a part of the required packages that will need to be installed. Some packages will be added or updated.
- **PhenoHiveStation.py**: This file contains the *PhenoHiveStation* class. In programming, a class is like a template used to create objects. It contains both data called attributes and functions called methods that describe the behavior of these objects. Our *PhenoHiveStation* class will therefore have, for example, attributes like *station_id*, *image_path*, *status*, and many others. It will also have functions like *get_weight*(), *save_photo*(), *measurement_pipeline*(), etc. Most of the changes that will be made will be in this file. For example, we will add functions and attributes.
- **config.ini**: This file contains several variables that are specific to a station and must be kept in memory even when the station is turned off. It is divided into different sections such as Station, InfluxDB, Paths etc. In these sections we find the variables:
  - **id**, which allows you to know which station is communicating with the database.
  - **token**, which allows you to identify yourself to the database.
  - The paths to the different folders where different data should be saved.
  - The different connection pins, for example, for the camera and the buttons.
  - **time_interval**, which defines the time between two plant measurements.
  - **calibration_weight** and **load_cell_cal** which are the reference weight for calibrating the scale and the factor for converting raw weight to weight in grams. This will be explained in more detail in the next section.
  - Parameters useful for image analysis.

  Later we will add pins for new sensors, change paths and addresses for a new database configuration and add variables.

- **image_processing.py**: This file contains the code for analyzing the image taken by the station. Its functions return a total measurement of the plant's skeleton, that is, a sum of the length of each leaf. In the rest of the thesis, we will add functions to this file to improve image capture.

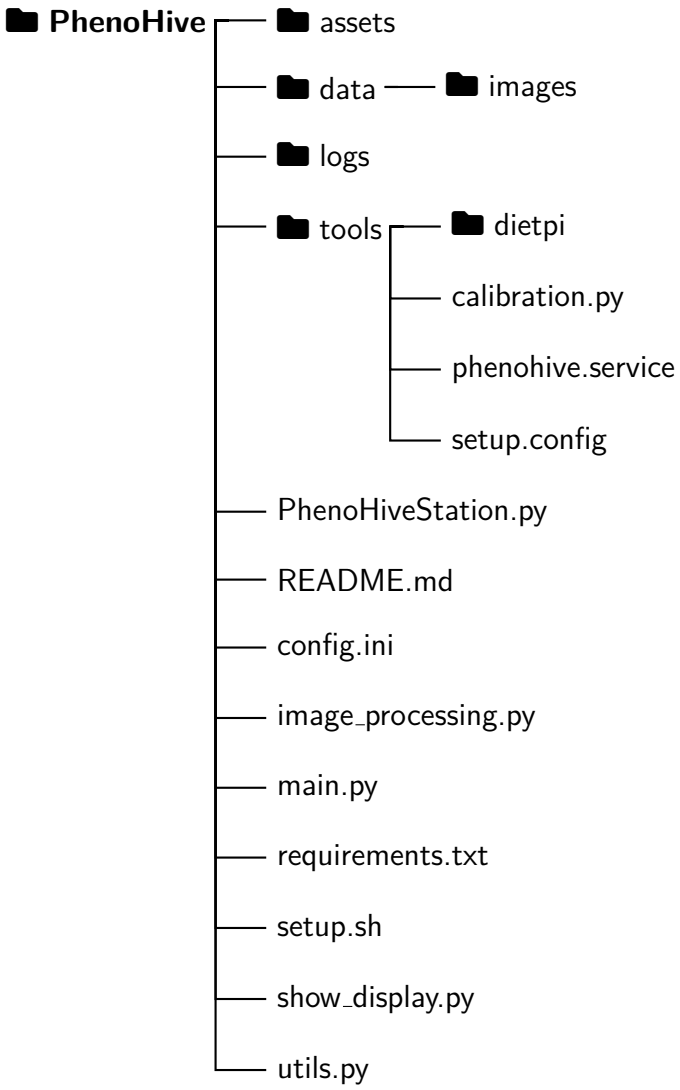
- **main.py**: This file is the heart of the station. It contains all the interface menus and interacts with the user via buttons or even with time to call up the functions used at that moment. Here again, we will add functions to adapt the station to changes.

- **requirements.txt**: This file contains the Python packages that need to be installed with pip[5]. Some packages will be added or updated.

- **setup.sh**: This file allows for the more manual installation of the station. It uses the **setup.config** and **requirement.txt** files to determine which packages to install on the Raspberry Pi. This file also allows us to enable the SPI or the camera. More details on this manual installation are available in the **README.md** file or in [2]. As we will be changing the Raspberry Pi, this file will need to be adapted. We will need to use this file to recreate a new image for the SD card.

- **show_display.py**: This file contains the functions that allow you to create the different screens to be displayed when the station is in operation. Adding new sensors goes hand in hand with displaying their measurement. This will make changes to this file.

[5]pip is a package manager used to install and manage packages written in Python [26].



3.2.4 Interface

In this section, we will show you how to use the station's direct interface. We will see how to calibrate the scale, how to correctly position the camera in relation to the plant, how to start taking measurements, and also how to find out the station's status.

To make it easier to understand the different navigation paths between menus, a diagram is shown in Fig. 3.7.

Menu

When you start the machine, you arrive at the main menu, which is titled "Menu." Each menu offers a maximum of two choices, which can be made using the two buttons on the right and the left. In this menu, you can choose to either go to the "Configuration" menu or start the measurements.

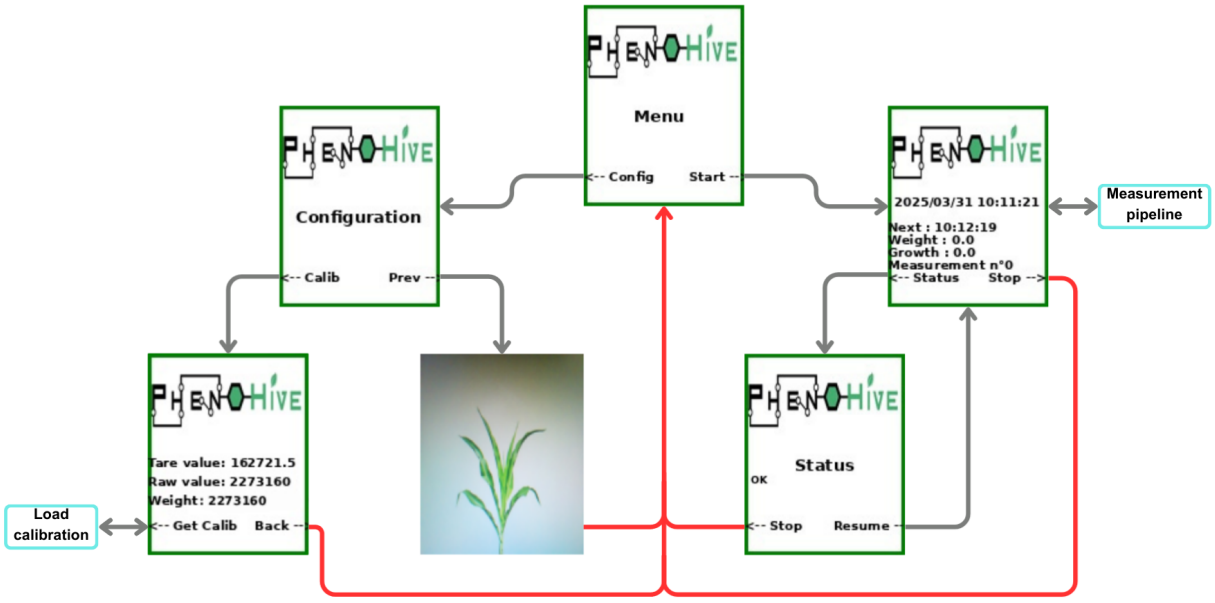


Figure 3.7: Diagram of possible navigations for each menu, inspired by Dylan Goffinet's thesis [2]

Configuration

From the main menu, we arrive at the "Configuration" menu. In this menu, we have two options: the one on the right takes us to the scale calibration menu, and the one on the left takes us to the "Preview" menu.

Calibration

This menu is used to calibrate a proportionality coefficient for the scale. More precisely, when the station measures the weight on the scale, it receives a raw value that is not in grams. In order for the user to read this weight in grams, we need to calculate a coefficient. We will then have:

$$weight\ in\ grams = coeff \cdot (raw\ weight - tare)$$

The procedure for calculating this coefficient is as follows: before entering the "Calibration" menu, that is, before pressing "Calib" in the "Configuration" menu, you must ensure that there is no object on the scale. This must be done because the program records the tare weight when entering the calibration menu. Once in the menu, you must then place a reference weight on the scale. By default, the reference weight is set to 1.5 kg. Once this is done, you can then press "Get calib". At this point, the program will calculate the coefficient from the following equation:

$$coeff = \frac{ref\ weight}{raw\ weight\ -\ tare} \tag{3.1}$$

To be sure of the calibration, it is possible to press "Get calib" several times to check if the *raw value* value does not fluctuate too much. This coefficient is then saved in the *config.ini* file. This avoids the need to recalibrate the scale each time the station is turned on. Similarly, the tare and the reference weight are saved in the same file. The screen displays the values of tare and raw weight, as you can see in the diagram Fig. 3.7.

Once the calibration is finished we can use the left button corresponding to "Back" which immediately takes us back to the main menu.

Later in this thesis, in section 4, we will change this menu to add an image capture calibration. The weight calibration will always be available but will change very little.

Preview

From the "Configuration" menu, we can access this menu. It displays the image captured by the camera in real time. This allows us to correctly position the plant so that it does not exceed the camera's field of view.

By pressing the right button, although there are no options visible on the screen, we can return to the main menu.

Measurements

The measurement menu displays various pieces of information. The first is the date and time. These are updated either when the station and therefore the Raspberry Pi are launched, or once a day. The second piece of information is the time at which the next measurement will be taken. By default, an image is taken every 60 seconds, but this time can be changed in the config.ini file. Next, we have the weight and growth values from the last measurement. Weight is displayed as a raw value, while the growth unit is pixels. Finally, we have the number of measurements taken since the last shutdown.

When the measurement menu is launched, the code will, at each instant, check that the current time has not passed the "next" time of the next measurement. If this is the case, the code then enters the "Measurement Pipeline". This measurement pipeline is represented by a diagram in Fig. 3.8.

The first thing the pipeline does is take a photo. This photo is then displayed. This allows us to check whether the plant is still correctly positioned within the camera frame. This photo will then be analyzed to determine, using a recognition algorithm, the total growth of all the plant's leaves in pixels. The calculated value is then displayed. The station then measures the weight detected by the scale and displays the raw value on the screen. Finally, the station sends all the collected data to the database via Wi-Fi. The pipeline is then finished and we return to the measurement menu until the next measurement.

You can stop the measurement at any time using the "Stop" button, which returns you to the main menu. It is also possible to use the other button to access the "Status" menu.

When we add new sensors later in the thesis, section 5, they will also be displayed in this menu as well as in the pipeline.

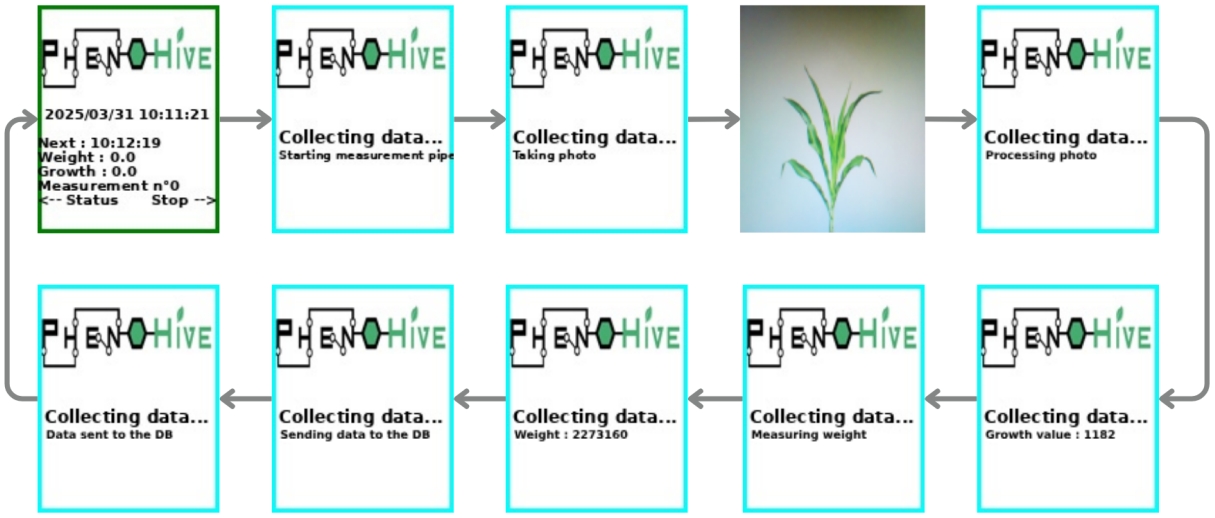


Figure 3.8: Measurement Pipeline

Status

This menu allows you to see the type of error that occurred when an error occurs. This is useful when the machine is running unattended for a long time. When there are no problems, the status simply displays ”ok” as shown in Fig. 3.7.

From this menu, we can either completely stop the measurements and return to the main menu, or return to the measurements menu, which will resume its operation as explained above.

The colored borders

As you may have already noticed, the screen borders are not always the same color. These actually correspond to the current status of the station and are of four types.

- Green borders correspond to a station that has no problems. This is what has been shown in the figures in most cases so far.
- When the outline is dark blue, it means the station encountered an error. These errors can be of several types. There may have been an error because the plant was misplaced and the code was unable to calculate the growth, because there is no good connection to the scale, because there is an error in the code (as long as this error is not fatal), because the camera is no longer working or is obstructed, etc. Some example images are shown in Fig. 3.9.
- When the borders are red, the station is working perfectly, with one exception: the database is not connected, and the station was therefore unable to send its data. Fortunately, the station can run by saving its own data on its SD card. Even if it can do this for quite a long time (228 days according to Dylan Goffinet’s thesis [2]), you still have to find the source of the problem, which is often a disconnection with the

Figure 3.9: Blue colored borders due to an error when taking the photo

Wi-Fi or an incorrect token used during initialization. An example of this border is shown in Fig. 3.10

Figure 3.10: Main menu with red borders when the station is not connected to the database

- The cyan blue borders appear when the station is in the measurement pipeline as we can see in Fig. 3.8

Thanks to these borders reflecting the status of the station, the user can see its status at a glance.

3.3 Minor additions and repairs

When the Phenohive station was received in September, the first step was to initialize it. This was done as explained previously, however, some bugs or inconsistencies were still present. For your information, when an error occurs in the code, the screen either doesn't respond at all and remains blank at startup if the error comes from the initialization or code syntax, or it freezes during use when the error is encountered.

The various debugging and additions are presented in the following sections.

3.3.1 Init parser

During his year, Dylan Goffinet worked on better organizing the project's files. To do this, he created a config.ini file to store the variables intrinsic to each station. To access this file and to be able to read and modify it, you need to use a parser. In our case, a Configparser is used. More precisely, it is: "the ConfigParser class which implements a basic configuration

language which provides a structure similar to what's found in Microsoft Windows INI files" [27].

The issue was straightforward: the parser had not been initialized. So the following line was added to the PhenoHiveStation.py code:

$$self.parser = configparser.ConfigParser()$$

3.3.2 Measurement displayed directly in grams

The code received at the beginning of the project displayed, in the measurement loop, the relative weight received by the scale. Since the relationship to calculate the weight in grams was already implemented, it seemed more judicious to me to display it in the measurement menu as well as the measurement pipeline.

Similarly, in the calib menu, a weight in grams was calculated. However it only displayed two values 0 or 1500 which is the default calibration weight. This is due to the formula:

$$coeff\ calib = \frac{ref\ weight}{raw\ weight\ -\ tare} \tag{3.2}$$

Where the *coeff_calib* value is directly reused, in the same transformed function, to calculate the displayed weight:

$$weight\ in\ grams = (raw\ weight\ -\ tare) \cdot coeff\ calib = ref\ weight \tag{3.3}$$

Which therefore gives:

- 0 if the tare is incorrectly initialized, i.e. with the weight already on the scale before entering calib mode, and therefore raw weight - tare = 0
- 1500 = *ref weight* for any weight placed on the scale and if the tare has been correctly carried out (weight placed after entering the menu). To be more precise, here is an example: a weight of 1000g is placed on the scale. Then, the coefficient is calculated by the following calculation (the numbers used are fictitious):

  $$coeff\ calib = \frac{1500}{1000 - 0} = 1.5$$

  This coefficient is then used to calculate the weight in grams:

  $$weight\ in\ grams = (1000 - 0) \cdot 1.5 = 1500$$

  This clearly shows that we arrive at a displayed weight of 1.5 kg regardless of the weight placed on the scale.

For display, we kept the value of *tare* and *raw weight* to which we added, as mentioned earlier, the weight in grams. This can be seen in Fig. 3.11.

In the end, the displayed data was not very informative or practical. So, an improvement was added instead. Once in the calib menu, the first three presses note "Get calib" which calibrates the scale with the 1.5 kg weight. From the fourth press, the "Get calib" button becomes a test button; another known weight can then be placed on the scale to verify that it is weighed correctly.

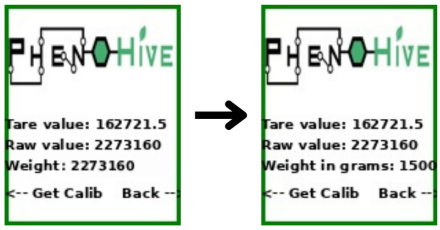


Figure 3.11: Added weight display in grams in the calibration menu

3.3.3 Running variable bug

The *running* variable stores the operational state of the station (1 = running, 0 = stopped). This state is saved in the *config.ini* file, allowing the system to automatically resume measurements after a reboot, for example following a power outage or software crash. It is also used in the measurement loop to determine whether the process should continue or stop when a user presses a control button, thus ensuring a safe and consistent shutdown of the measurement process.

The problem with the code received was that this variable was not correctly written to the *config.ini* file or in the main loop. This caused an infinite loop in the measurement menu from which it was no longer possible to exit. In fact, in the main menu, we have a condition, checked at each loop, which tells us: *if running is 1 then enter the measurement loop*. The problem then is that, once the value of running was set to 1, it was never reset to 0 again. This problem was solved quite simply by adding a *running* attribute to the *PhenoHiveStation* class and by correctly updated in the *config.ini* file.

3.3.4 Display adjustment

Text and writing

As we can see in Fig. 3.9, some displays are sometimes incorrectly adjusted. This is not a fatal error for the station, but for the user it is still more pleasant to be able to read the sentences completely and not have certain displays cut off. A complete correction of all displays has been carried out, that is to say of all menus and submenus. Fig. 3.9 shows the three main display errors:

- Parts of words/measures, as well as the arrows corresponding to buttons (often the right one), are cropped and/or covered by the colored border. An example can be seen on the first screen of the figure.
- When the sentences to be displayed in ”Collecting data...” are too long, they go outside the display box. An example can be seen on the second screen of the figure.
- The display of the ”Status” menu is somewhat chaotic. An example is shown on the third screen and in Fig. 3.12.

Figure 3.12: Improved display: in "Collecting data" with a line break (left) and in "Status" with date formatting plus a line break (right)

The first type of error was simply fixed manually one by one. For exceeding measurements, rounding has also been added to the displayed values. For errors two and three, the changes were made directly in the *show_collecting_data*() and *show_status*() functions, which now take into account the length of the sentence to be displayed and the trimming if necessary. For the "Status" menu, the PhenoHive logo has been removed to save space and the date has been put in a more visually pleasing format.

Status border color

We have a second display issue that isn't critical but is best fixed to maintain consistency. The code borders, until now, weren't actually the desired color. In Dylan Goffinet's thesis [2], we are told that the colors for each status are:

- : No error
- : No error but not connected to the database
- : Error in the measurement pipeline
- : Measuring in the measurement pipeline

This is also what is referenced in the code. However, as we saw in the 3.2.4 section, the colors displayed on the screen are different.

The problem here was that the RGB (Red Green Blue) code was reversed and the colors should be given in BGR. Once this correction was made, the statuses were displayed perfectly.

3.3.5 Multithreading of initialization

To improve the performance of our station, a choice was made to move from a Raspberry Pi Zero W to a Raspberry Pi Zero W 2. The justification for this choice will be made in the next section but for the moment we can only remember that we are moving from a Raspberry with only one core to a Raspberry with four. Thanks to this improvement, we gain access to a very useful capability, which is multithreading. Multithreading is a programming technique that allows multiple tasks to be executed simultaneously within a single program by dividing its execution into several independent "threads." Each thread

can then perform a different part of the work, which optimizes the use of processor resources and improves the responsiveness of a code.

In the current state of the code, the station takes approximately 1 minute and 55 seconds to initialise. This means that between plugging in and using the station, the user has to wait almost two minutes. Thanks to multithreading, we were able to divide the initialization into several functions: *init_display*, *init_influxdb*, *init_camera_button*, *init_load*, and *init_data*. These six functions are therefore handled one by one by the four cores, which will first execute the first four functions. Then, once a thread has finished its work, it will take another function, until all six have been executed.

Finally, thanks to this small improvement, the station initializes in 52 seconds. This means we have reduced the initialization time by half.

3.3.6 Error from the camera

At the end of the research dedicated to the thesis, we ran the station for a long period of time, lasting about a week. Later, we will talk about this period in other sections. For now, we will explain the resolution of an error encountered at that time.

During the measurement, an error was revealed to be coming from the camera. The problem with this error was very worrying because it wasn't a simple one-time loss of measurement, which, while annoying, wouldn't have been so serious. Indeed, the error encountered caused the entire station to freeze, which then had to be unplugged and reconnected. This was critical because the station's initial goal is to operate with as little human intervention as possible.

To avoid this error, a simple *try/except* wasn't enough, as it occurred at a lower level than the project code, namely in *libcamera*, the library/tool responsible for capturing images and videos on the Raspberry Pi. We therefore had to create, from the Python code, a parallel process with a predefined timeout. So, if an error occurs that would normally cause a crash, the program terminates that process and then continues executing the code.

3.4 Initialisation and user manual

Since the station is intended for educational purposes, we have created a manual for students. In it, we cover several sections of this chapter, such as initialization and the explanation of the interface. These explanations have been reduced to the essentials of what the student needs to know. The information has also been adapted to reflect all the changes that will occur in this thesis. This manual is available in the appendix D.

Chapter 4

Improved image capture

In this section, we will discuss how to improve image capture. While weight measurement seems to be going smoothly, growth measurement, which is captured through images, still has some deviations. We will therefore start by upgrading our system to a newer version of PlantCV. We will then discuss the different parameters used by the growth calculation algorithm, and finally, we will add a preprocessing to remove shadows from an image.

4.1 Updating PlantCV to a newer version

4.1.1 Comparison of different software packages and boards

So far, we have PlantCV version 3.14.3 installed on our workstation. However, as already stated in [2], this version is limited and does not allow us to fully exploit PlantCV's capabilities. Due to the use of a Raspberry Pi Zero W, updating PlantCV to its current version (4.6) is currently impossible. Therefore, we questioned the possibility of changing the motherboard or the software used.

We first compared different software packages based on the following criteria: analysis accuracy, ease of use, required hardware resources, and flexibility. All these criteria were used to complete a table found in the appendix E. Finally, among the candidates, we decided to retain PlantCV because it stands out for its specialization in processing plant traits. Unlike general-purpose tools like ImageJ or OpenCV, it offers functions already adapted to the project's needs. It's also lighter and more customizable than deep learning-based solutions like DeepPlantPheno or Phenotiki, and it remains free and well-documented. It's also easy to use thanks to its use of Python.

Once the software has been chosen, we can focus on the potential motherboard upgrade. Here again, we have created a comparison table of different circuits available in the appendix E. The comparison criteria are as follows: price, computing performance, ease

of integration into the existing system, community support available, and native Wi-Fi or Bluetooth integration. Finally, for this project, we need an inexpensive circuit that's easy to use and powerful enough. The Raspberry Pi Zero 2W is therefore a good choice: it's small, inexpensive, and works well for capturing images and performing simple processing. Other circuits like the Jetson Nano are more powerful, but also more expensive and complicated to use. The BeagleBone or Odroid require more adaptation of the existing setup and are expensive too. The Pi Zero 2 W also has the advantage of having Wi-Fi, many examples available online, and it works well with the software we use, like PlantCV.

To be more precise in terms of price, we can note that between the Raspberry Pi Zero W which is at 19.95€ and the Raspberry Pi Zero 2 W which is at 21.95€ there is only a difference of 2€. In addition, the Zero 2, as already mentioned in a previous section, has 4 cores where the Zero has only one.

4.1.2 Adaptation of software initialization

Since the Raspberry Pi Zero 2 W is simply a more powerful upgrade of the version used until now, the Raspberry Pi Zero W, the physical connections remain unchanged, which is very convenient. However, some changes need to be made to the package installation. This is unfortunate, but we can't simply update PlantCV by reusing the installation files provided in previous thesis [2] as is.

The first changes we will make are in the *requierements.txt* file, which becomes:

```
imageio==2.37.0
configparser==7.0.0
influxdb-client==1.44.0
hx711==1.1.2.3
pandas==2.2.3
adafruit-gpio==1.0.3
picamera2==0.3.18
opencv-python==4.7.0.72
plantcv==4.6
mizani==0.9.0
plotnine==0.10.1
```

In this file, we changed three things. First, we upgraded the PlantCV version to 4.6 instead of 3.14.3. This is the basic purpose of the change. Second, since PlantCV uses pandas and the previously used version (1.5.1) was no longer compatible, we changed it to version 2.2.3. Finally, the new version of PlantCV now uses a package that we didn't yet have on the workstation. This is imagio, which we're adding to the list with its most recent version.

In addition, the *setup.sh* file must be modified to ensure no camera issues, and the *config.txt* file must be modified to adapt to the new 64-bit ARM architecture and GPU.

Finally, after these changes, the installation of the new Raspberry software can be done using the *setup.sh* file in the same way as explained in [2].

4.2 Calibration of the parameters used by the growth calculation algorithm

4.2.1 Presentation and explanation of the parameters

The image analysis pipeline that makes up the agorithm we use has several parameters that can be modified and then give us better or worse results. To better understand them, we will briefly review this algorithm. This is explained in more detail in the two previous reports [1, 2].

An illustration of the analysis pipeline is given in Fig. 4.1. So, as the first step, we convert a color photo to a grayscale image. Then, the plant's contours are detected using a Canny filter. These contours are then filled and the skeleton calculated. Finally, the length of this skeleton in pixels will be the final measurement of growth.

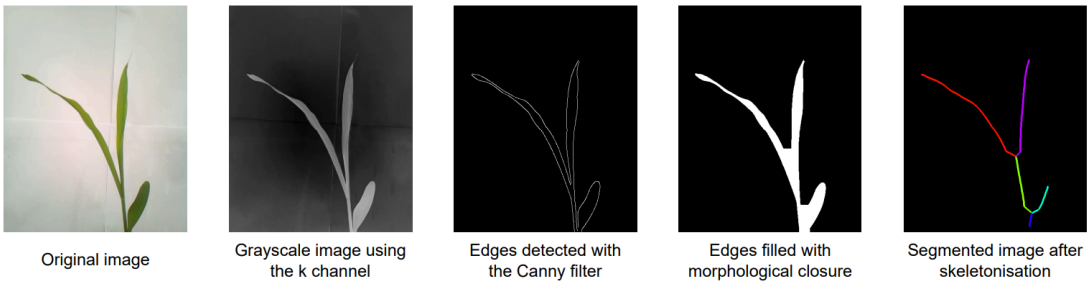


Figure 4.1: Pipeline of image analysis used for growth [2]

In the steps mentioned above, two parameters stand out and greatly influence the pipeline results. First, we used **sigma** with the Canny filter. Simga is "Optional standard deviation of the Gaussian filter" [28]. It is actually used to control the sensitivity of edge detection during this step. A low sigma value allows for the detection of fine details but may introduce noise, while a higher value smooths the image more and allows only the main edges to be retained or even no edges at all if sigma is too high. Adjusting sigma therefore allows for the optimization of skeleton detection based on the quality and contrast of the image.

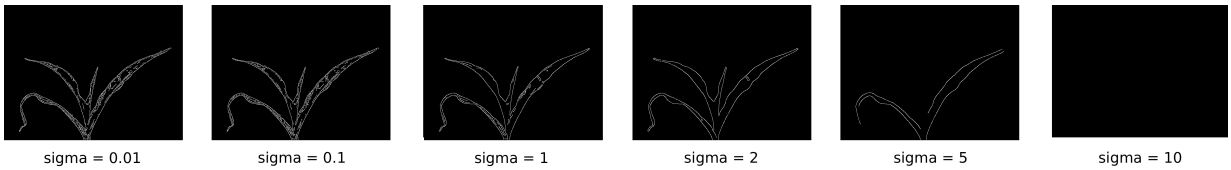


Figure 4.2: Effect of increasing the sigma parameter in the algorithm

The second parameter is the **kernel_size** used for filling. This parameter actually determines the size of the kernel used for the morphological closure operation (cv2.MORPH_CLOSE) that fills small holes or discontinuities in the detected edges. A larger kernel size allows for larger gaps to be closed, but may also smooth or distort small details, while a kernel size that is too small will not close enough gaps.

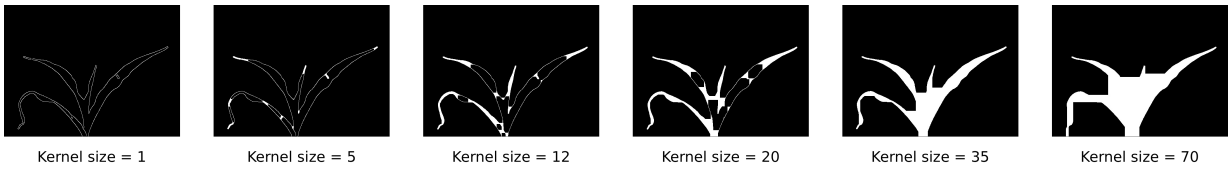


Figure 4.3: Effect of increasing the kernel_size parameter in the algorithm

4.2.2 Addition of an algorithm for calculating optimal parameters

Given what was just presented in the previous section, it seems obvious that finding an optimal combination of these two parameters is essential for detecting the plant skeleton. This combination must therefore respect a compromise for each extreme of each parameter.

While in the code, until now, we had parameters set to 2 for sigma and 20 for the kernel size, here we decided to find the best parameters almost automatically. The code works as follows:

- A photo of the plant is first taken by the station. This photo is stored in memory in the station but must be retrieved on our computer for further processing. To do this, we use the command below:

  ```
  scp root@192.168.1.15:~/PhenoHive/data/images/img_calib.jpg
  C:\Users\path to folder
  ```

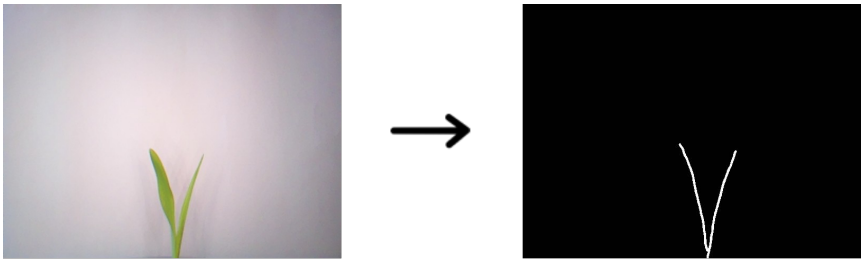
  In this command, we first have the IP address of the station. This can be easily found with the nmap tool [1]. The protocol is explained in [2]. Next we have the path to the image taken by the station, named *img_calib.jpg*. Finally, we have the path to the folder on our computer where the image *img_calib* will be copied.

- Once the photo is retrieved, we will manually create a reference skeleton that will be used for the algorithm. This can be done using photopea [2] as explained in more detail in the appendix F, which is a free, online equivalent of Photoshop. The skeleton should then look like what is visible in Fig. 4.4. In the same way as before, we will send this image of the reference skeleton back to the station using the following command:

  ```
  scp C:\Users\path to folder\skeleton_ref.jpg
  root@192.168.1.15:~/PhenoHive/data/images/
  ```

- Once these prerequisites are met, we can actually run the algorithm. It will start by creating all possible combinations of the two parameters from two lists created from the initial values (we will explain how a little further on). That is, for example, for the values of sigma = 2 and kernel size = 20 (the values when the station was first launched), we will have the lists: [0.05, 0.444, 0.888, 1.333, 1.777, 2.222, 2.666, 3.111, 3.555, 4] and [15, 16, 17, 18, 19, 20, 21, 22, 23, 24].

[1] https://nmap.org/
[2] https://www.photopea.com/



img_calib.jpg skeleton_ref.jpg

Figure 4.4: Manual transformation of the image taken for calibration, img_calib, into a reference skeleton, skeleton_ref

- For each combination, we will then calculate a similarity index. The Sørensen similarity index, Sørensen-Dice coefficient or Dice similarity coefficient (DSC) is a measure used to compare the similarity between two sets [29]. It is defined as:

$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$

  where A and B are the two sets to be compared. The index varies between 0 (no similarity) and 1 (identical). It is perfectly suited to image processing to evaluate the overlap between two binary masks. In our case, we will calculate, for each parameter combination, a skeleton that we will then compare with the reference skeleton constructed earlier manually. Each of these comparisons will then give a DSC. This DSC is, here, calculated as follows: $|A|$ is the number of white pixels on the image of the calculated skeleton while $|B|$ is the equivalent for the reference skeleton. $|A \cap B|$ is the total number of white pixels placed in the same place on both images. Finally, we just need to take the largest DSC to find the two parameters that gave the best result. Given the very large number of combinations, we used multithreading, made possible by the RaspberryPi Zero 2 W, to speed up the calculation.

- The final step is to repeat the process to refine the result. Since sigma is a decimal point, its value can be more precise than just the values present in the first list. The list is then created based on the number of tests performed so far. The piece of code giving the list is:

```
spread = sigma / calib_test_num
sigma_values = np.linspace(sigma - spread, sigma + spread, num=10)
sigma_values = np.clip(sigma_values, 0.05, 5)
```

  where *calib_test_num* is the current calibration test number $(1, 2, 3, \dots)$.

  The refinement works as follows: after each calibration test, the sigma value giving the highest DSC is kept as the new reference. This value becomes the center of the search range for the next test. The range is progressively reduced according to:

$$\text{spread} = \frac{sigma}{\text{calib\_test\_num}}$$

For example, if the best *sigma* after the first test is 1.5, the second test will explore values in the interval $\left[1.5 - \frac{1.5}{2},\ 1.5 + \frac{1.5}{2}\right]$ using 10 equally spaced points. This process is repeated at each iteration, with the interval narrowing around the best value, thus increasing precision. Values are clipped between 0.05 and 5 to avoid unrealistic values. The same logic is applied to the kernel size, except that it must remain an integer; in that case, we simply take 5 values below and above the current best kernel size. The best DSC from the previous step is always compared with the new one to decide which to keep.

As mentioned earlier in section 3.4, an initialization and user manual has been produced. In it, we explain, without going into detail, the concrete steps mentioned above that allow students to calibrate the parameters. This is available in the appendix D.

4.2.3 Integration of calibration into the interface

The addition of this new feature goes hand in hand with a modification of the interface. These changes are simple, but to better follow them, an illustration is given in Fig. 4.5. When we arrive in the **Configuration** menu, we can now navigate to the **Calibration** menu, which includes the calibration of the weight and image parameters. Previously, there was only the weight to calibrate at this location. From this new menu, we navigate directly to the calibration menu for the sigma and kernel size parameters. In this menu, we start by pressing the left button to take a photo. It is then at this point, after the photo has been taken, that we can retrieve it on the computer and create the reference skeleton that will be sent back to the station. When this is done, we press again the left button to start the parameter calculation. When it is finished, we return to the menu's starting screen where the values have been updated. A second loop can then be launched: if the station and the plant have not moved, it is sufficient to take another photo and immediately restart the calculation; on the other hand, if the plant or the station have moved (which is not problematic), it is necessary to take a new photo, generate a new reference skeleton, then launch the calculation. The display will then show "Calib 2", "Calib 3", etc.

4.3 Image preprocessing using shadow detection

The detection of the plant skeleton is at this stage quite satisfactory. We will analyze the results in the next section. However, there remains a problem that was raised by Dylan's thesis [2]. These are actually shadows that, when they appear, affect growth measurements because they are mistaken for the plant by the pipeline.

4.3.1 Chromacity-based method

Although this problem is not common and can easily be avoided by placing the station and the plant in a good environment from the beginning, we still added an image preprocessing before the skeleton detection to remove shadows. The method used to achieve this is based

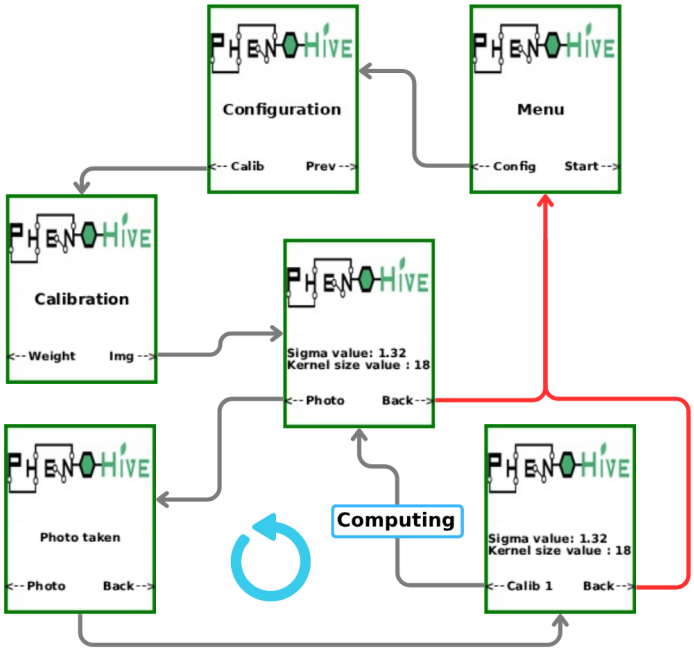


Figure 4.5: Diagram of the interface integrating the calibration of the image capture parameters

on the research paper *Shadow Detection: A Survey and Comparative Evaluation of Recent Methods* [30]. More specifically, in this paper, we will use the *Chromacity-based method*.

The chromaticity-based method is based on the idea that when an area is covered by a shadow, its light intensity decreases, but its color (chromaticity) remains virtually the same. For example, a green pixel becomes dark green under the shadow, but retains the same shade of green. To exploit this property, a color space that separates intensity and chromaticity better than RGB space is often used, for example, HSV. In HSV space, hue (*Hue*) indicates color, saturation (*Saturation*) measures the purity of the color, and value (*Value*) represents intensity. A pixel p is then considered to belong to a shadow if the following three conditions are met:

$$\beta_1 \leq \frac{V_F(p)}{V_B(p)} \leq \beta_2 \tag{4.1}$$

$$S_F(p) - S_B(p) \leq \tau_S \tag{4.2}$$

$$|H_F(p) - H_B(p)| \leq \tau_H \tag{4.3}$$

where $V_F(p), S_F(p), H_F(p)$ are the components of pixel p in the current image (*Frame*) and $V_B(p), S_B(p), H_B(p)$ are those of the corresponding pixel in the background image (*Background*). The parameters $\beta_1, \beta_2, \tau_S, \tau_H$ are empirically determined thresholds. These three conditions can be translated as follows: the pixel's intensity is weaker than that of the background, but not too weak, its saturation has changed little, and its hue is very close to that of the background.

4.3.2 Implementation in code

Now that we have the basics of the method we're using laid out, we need to implement this approach into our image processing pipeline. But before that, we need to integrate a new feature into our workstation: capturing the background.

So we added this feature to the *Preview* menu. While the right button used to exit the menu, the left button wasn't associated with any event. Now, when it's pressed, a *background.jpg* photo is saved in the station's *image* folder and can be used later. The *Preview* menu is particularly suited to this function, as it directly displays the area captured by the camera. To obtain a correct background, simply take a photo of the background, without the plant or any other objects.

Once this step is completed, we integrate the function *remove_shadows_hsv* into the code, which takes an RGB image of the plant as input and successively applies the following steps:

- **Reading and adjusting background luminance**: Before any comparison, the background image is loaded and passed to the *match_luminance* function. This function locates the brightest area in the current image and adjusts the background luminance so that it locally matches that of the image. This reduces the impact of global lighting variations between shots, making the color comparison more reliable.
- **Conversion to HSV space**: The current image and the (adjusted) background image are converted to HSV. The H, S, and V components of the two images are separately retrieved.
- **Shadow Pixel Detection**: For each pixel, we calculate the intensity ratio V_F/V_B, the saturation difference $S_F - S_B$, and the circular hue difference[3] $|H_F - H_B|$. Pixels simultaneously satisfying the three conditions (4.1), (4.2), and (4.3) are marked as belonging to a shadow.
- **Shadow Mask Post-Processing**: A Gaussian blur and thresholding are applied to clean the binary mask, followed by a dilation step to smooth the edges of the detected areas. This avoids overly abrupt artifacts when replacing these areas with the background.
- **Replacement and Smoothing**: If the number of detected pixels exceeds a certain threshold (here 20 000), the shadow areas are replaced by the corresponding pixels from the background image. A smooth transition is achieved through local blurring and bilateral filtering to preserve details and avoid a visible "patch" effect.

Ultimately, this step produces an image that is virtually shadow-free while preserving the plant's original structure and color. This corrected image is then used for contour detection and skeletonisation, as described earlier in this chapter.

[3]The circular difference is the minimum difference between two values located on a periodic space. For example, between 350° and 10°, the circular difference is 20° and not 340°.

4.3.3 Results

In order to judge the results of the shadow detection algorithm, we created two series of photos: the first, of around twenty shots, with shadows created artificially using a spotlight, and the second by placing the station in an environment where many shadows could appear naturally.

Some photos are shown below, Fig. 4.6. In these we can see some examples that show that the preprocessing works without being infallible. The first line shows that the preprocessing can sometimes not change the result, the second and third show that it works even with large shadow areas and the fourth shows that shadows that may seem light still sometimes strongly impact the skeleton which can then be corrected by the preprocessing. Finally, since the pre-processing is not perfect, it may happen that it does not completely remove the shadows, as in the fifth row where the background shadow, particularly severe, blended too much with the base of the stem. The pre-processing can also sometimes be too aggressive on the finer areas of the plant, as in row six where part of it was lost. However, we will see in the following section that post-processing the data will reduce these undesirable effects.

To quantify the difference between using pre-processing and not using it, we took all the photos containing shadows (1055 in total) and counted the number of skeletons that were significantly different from reality. Without pre-processing, we obtained 659 unsatisfactory skeletons, compared to only 182 after applying it. This gives a ratio of correct skeletons of 35.5% versus 82.7%, proving the usefulness of pre-processing even though it is not completely effective.

4.4 Long-term measures

First period: a week

As explained in the 3.3.6 section, we carried out a measurement over approximately one week. The growth measurement is visible in Fig. 4.7. In this graph, the initial blue curve corresponds to the raw measurements. We observe numerous peaks corresponding to outliers, with variations of up to 50 to 60 pixels from one minute to the next, which is relatively significant.

To address this, we have implemented two improvements:

- To smooth the curve, we calculate the average of the last 9 measurements and the the actual value. Then we take this value as final value.
- In the case of an outlier, defined here as a difference greater than 20 pixels from the previous measurement, this value is replaced by the previously saved value.

With these two improvements, we get the orange curve in Fig. 4.7. While this curve is still not perfectly smooth, it has improved significantly. We could add more values to the average or decrease the threshold of outliers, but we also don't want to lose the information provided by each measurement.

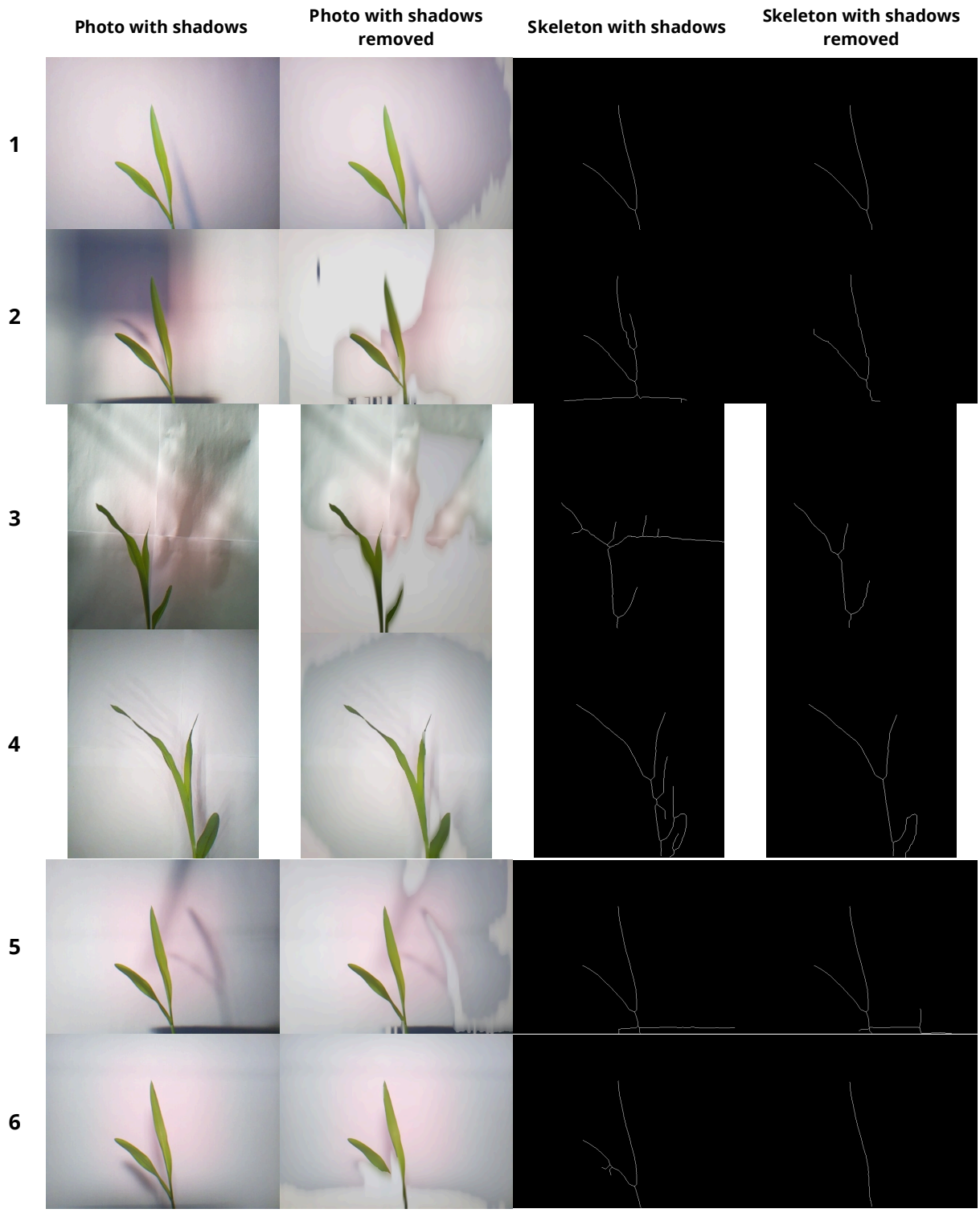


Figure 4.6: Different photos and their skeletons with or without the use of shadow removal pre-processing

One last strange thing we can notice in the growth curve is that it increases at first and

then, from the middle, begins to go down as if the plant is getting smaller. In fact, the problem here comes from the plant itself and not from the station. As we can see in Fig. 4.8, the plant gradually turned towards the light source which was a window on the left. After a while, one leaf then passed in front of the other which altered the measurement. This therefore illustrates the fact that it is important to choose the environment in which the station and the plant are placed carefully.

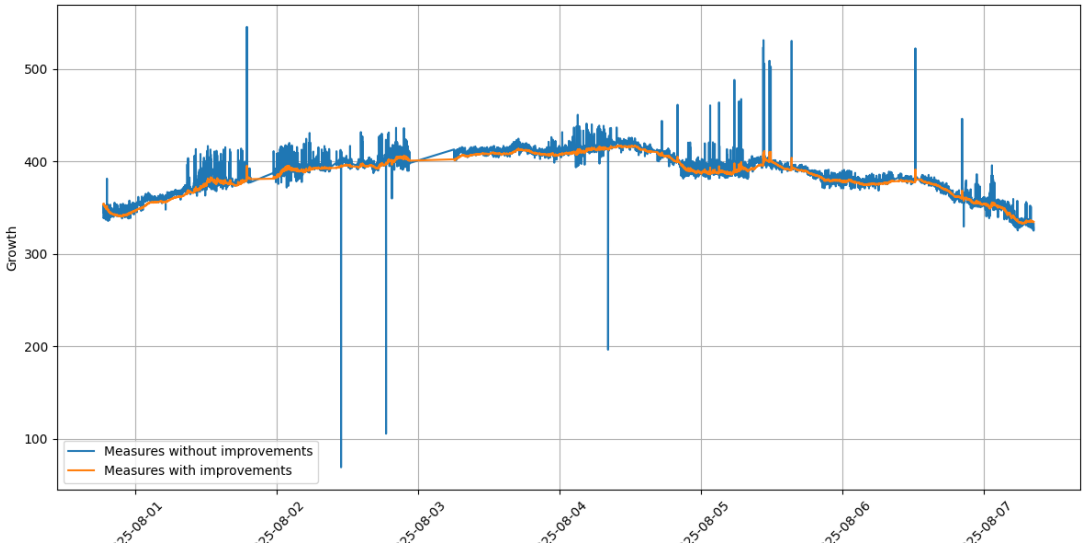


Figure 4.7: Growth measurement during the first long period

Figure 4.8: Plant evolution during the measurement. On the left, the first photo taken, in the middle, a photo taken after 4 days and on the right, the last photo taken.

Second period: a little over three days

Unable to obtain a satisfactory growth curve, we restarted the station with another maize plant in another sunnier room. The experiment lasted only a little over three days, but it showed rapid growth of the corn. The result is shown in Fig. 4.9. This graph shows almost linear growth, with the exception of one particular point. This rapid increase is completely normal: it corresponds to the repositioning of the station to keep the plant in the camera's field of view. As the students conduct the experiment, this type of readjustment will likely occur several times. It is important to keep in mind that the measurement expressed here is relative, in pixels, not centimeters. During the experiment, shadows appeared several times. The times when this happened are marked in red on the graph. We can see that

the image preprocessing by removing shadows worked quite well because there was no significant increase in these areas.

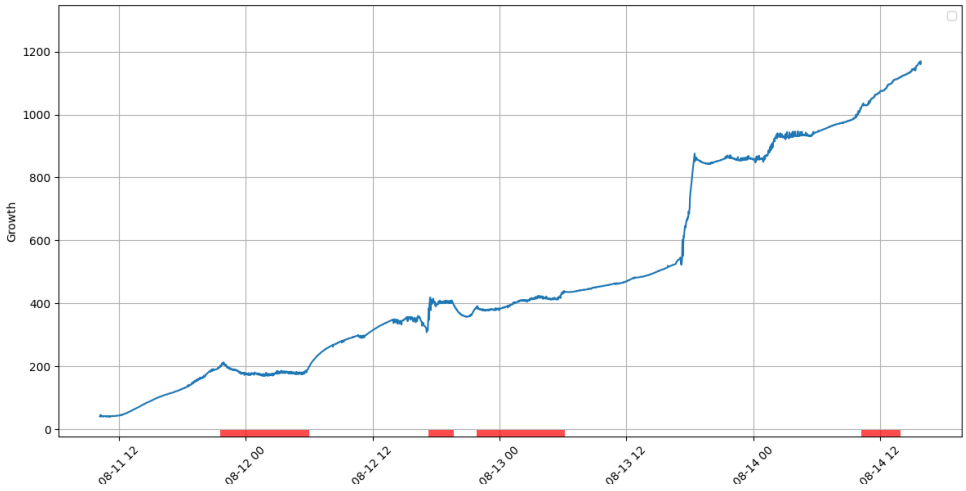


Figure 4.9: Evolution of plant growth during the second long-term measurement period

Chapter 5

New sensors

We currently have two sensors on the station: the scale and the camera. Since the weight measurement was fully functional, no major additions or modifications were made to this sensor during this thesis. The image capture was explained in detail in the previous chapter. We will therefore begin this section by explaining the process followed when adding two new sensors, one for soil moisture and the other for light.

5.1 Humidity sensor

5.1.1 Interest of the addition

Regarding the addition of a soil moisture sensor, it is interesting because it would allow for a better understanding of the water losses measured by the scale. Indeed, the latter records both plant transpiration and soil evaporation. Thanks to this sensor, it becomes possible to identify phases where soil evaporation dominates (wet soil) or decreases (dry soil), and thus to more accurately estimate the water exchanges. This sensor also provides advantages for managing experimental conditions, particularly by enabling the control or simulation of water stress[1]. Finally, it would allow for a better interpretation of growth dynamics by linking it to the soil's water status.

5.1.2 Overview

The humidity sensor chosen is a resistive sensor. It was chosen for its low cost of 1.44€. This sensor consists of two parts, which can be seen in Fig. 5.1.

The first is the resistive probe itself. This part is inserted into the ground and acts as a resistor that varies depending on the humidity in the pot. The second part of the

[1] Also called "water scarcity" or "water shortage" in extreme cases, water stress occurs when available water is insufficient or of poor quality compared to needs [31].

Figure 5.1: Soil moisture sensor [7]

sensor is a module composed of input and output connections, a resistor, a potentiometer, an LM393 comparator, and LEDs. Its operation is actually quite simple and relies mainly on a voltage divider. A voltage divider is a basic electrical circuit that generates a voltage lower than the supply voltage using two resistors. Its schematic is shown in Fig. 5.2. The output voltage is given by the following formula:

$$V_{out} = V_{in} \frac{R_2}{R1 + R2} \tag{5.1}$$

The output voltage therefore depends on the resistors. If we then have a variable resistor, our output voltage will then vary according to it. It is precisely this variation that interests us and which allows the module to transform a resistance value into a voltage that can be received by the raspberry pi. As said above, the module also includes a potentiometer and a comparator. These are used to define the digital output D0 of the module. The comparator compares the voltage received from the divider with a threshold voltage defined by the potentiometer. This system is useful for applications such as watering up to a desired humidity threshold but is not useful in our case because we will only use the analog output A0 of the module allowing a clear measurement of the humidity.

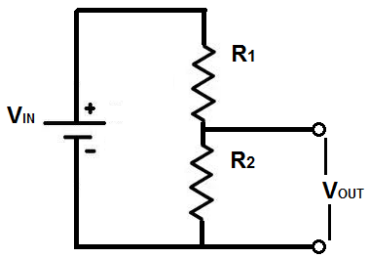


Figure 5.2: Voltage divider [8]

5.1.3 Integration into the station

To integrate our new sensor into the station, several changes are necessary. We will walk through the different steps involved.

Connections

The sensor is connected to the Raspberry Pi via three connectors:

- The input voltage: Vcc
- The ground: GND
- The analog output voltage: A0

It may seem simple to connect to the Raspberry Pi, but some questions may still arise.

The first problem encountered concerns the rapid oxidation of the sensor when exposed to humidity. However, this oxidation only occurs when a current flows between the two sensor pins. To preserve its lifespan, it is therefore best to power it only occasionally, during measurements, to avoid direct current flow.

On a Raspberry Pi, two supply voltages are available: 5V and 3.3V. In addition to these power lines, several types of connections are possible, including GPIO (General Purpose Input/Output) pins, which allow signals to be sent or received in a controlled manner. We will therefore use one of these GPIOs to power the sensor only during measurements, then disable it immediately afterward. Since the GPIOs on a Raspberry Pi are 3.3V, we will use this voltage as the power supply.

The second problem is that the Raspberry Pi is not capable of reading an analog voltage. To solve this problem, we need to use an ADC (Analog to Digital converter). An analog-to-digital converter is an electronic component that transforms an analog voltage, between GND and Vcc, into a binary value. This digital value is encoded on a certain number of bits, ranging from 0 (corresponding to 0 V) to $2^n - 1$, where n is the number of bits in the ADC. The higher this number of bits, the more accurate the conversion of the analog voltage into a digital value will be. In our case, we chose the MCP3008 ADC [32]. This component costs 2.72€[2] and has the particularity of having 8 channels. These channels correspond to the analog voltage inputs that the ADC can measure. Thanks to the code, it is possible to select the channel that we want to read. It is not possible to perform multiple readings simultaneously, but this is not a constraint for our project, because the measurements are performed sequentially via a pipeline. Thanks to these 8 channels, we can then connect other sensors with an analog output. This will be useful for the light sensor presented later in this chapter and will also facilitate the addition of other sensors in the future. This ADC has 10 bits and can be powered between 2.7 and 5.5V. Due to the choice made previously, we will use 3.3V here.

Like the display, the ADC connects to the SPI. The SPI is a bus shared between different devices. To use the SPI and avoid conflicts, each device has a Chip Select (CS) input. When a device's CS is enabled, the device, in our case the display or the ADC, knows it can use the SPI and start sending data.

All connections are shown in a connection diagram in the appendix A.

[2] `https://shop.mchobby.be/fr/ic-cms/2143-mcp3008-convertisseur-adc-a-8-canaux-soic-16-3232100021433.html`

Software adaptation

Once the sensor is correctly connected, we need to adapt the code accordingly.

The first thing to do is to use the MCP3008. To use it, we import a library *Adafruit-MCP3008* [33]. This import also induces, in the *requierements.txt* file, an addition of the following line:

$$adafruit - mcp3008 == 1.0.2$$

As a reminder, this file is used to download the various libraries during the installation of the station. Using this library, we can first initialize the ADC via the following line of code:

$$self.mcp = Adafruit\_MCP3008.MCP3008()$$

In the function we can specify the use of SPI and CS. The MCP3008 is then an attribute of the PhenoHive station that can be called at any time. This is what we do when taking humidity measurements in the measurement pipeline via the following line of code:

$$analog\_voltage = self.mcp.read\_adc(self.HUM) * (3.3/1023.0)$$

In this line, we first find the function *read_adc()*, which requests to read the value from the channel *HUM* corresponding to the channel where the humidity sensor is connected. Finally, we perform an analog to voltage conversion using the fraction $\frac{Vcc}{2^n-1}$.

Once the analog voltage is found, we apply a function:

$$\%\ of\ humidity = f(analog\ voltage)$$

This transfer function allows us to have a more meaningful value when discussing humidity. This function will be explained in more detail in the next section about calibration.

To inform the user, the interface has also been changed. First, during the measurement loop, we now display the humidity at the time it is measured. The adapted loop is shown in Fig. 5.3. Then, in the measurement menu, the humidity is now displayed and updated after each measurement loop completed. An example is shown in Fig. 5.4.

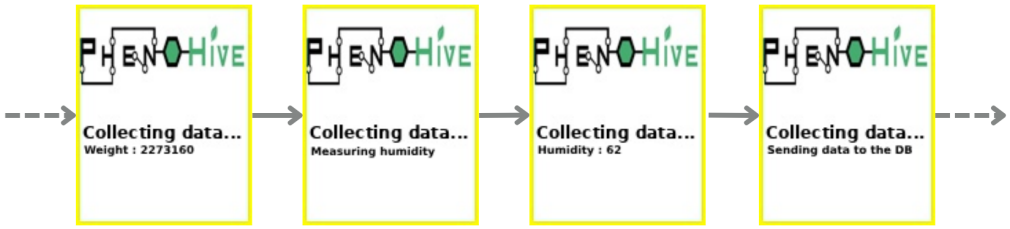


Figure 5.3: New display including humidity in the measurement pipeline

The final modification involves saving the new data and sending it to the database. After adaptation, we can modify the dashboard on Grafana, so that it allows us to know the humidity level at any time. To modify the dashboard, you can consult the appendices present in [2].

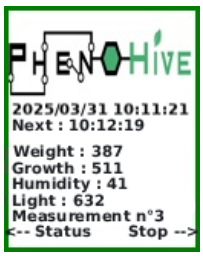


Figure 5.4: Measurement menu to which humidity and light measurements have been added

5.1.4 Sensor characterization

Transfer function

The goal of this section is to derive a transfer function relating the ratio between the mass of water and the mass of the soil. This is the function we will use in the code, as discussed earlier.

To find this function, we need to set up an experimental protocol. It is as follows:

1. A known mass of soil is taken initially. Here we arbitrarily use 28g.
2. Between each measure, we add water to the soil to humidify it. The quantities used are in accordance with the table below. To achieve 100% humidity, we only add water.

Desired rate [%]	Mass of the soil [g]	Water mass [g]	Added water [g]
0	28	0	
10	28	3.2	3.2
20	28	7	3.8
30	28	12	5
40	28	18.7	6.7
50	28	28	9.3
60	28	42	14
70	28	65.4	23.4
80	28	112	46.6
90	28	252	140
100	0	∞	∞

Table 5.1: Water and soil masses used to define the transfer function

3. For each measurement, a pot is filled with moistened soil. It is then compacted and the humidity sensor is fully inserted into the pot so that the metal areas are completely covered.
4. Finally, a Python program is launched to collect the measurements. This program is very simple and takes one measurement per second and then prints it. For each humidity level, 50 measurements were taken.

Using this experiment, we can find a function that gives the measured voltage as a function of the humidity. To do this, we average the measurements at each humidity level. This graph is shown in Fig. 5.5 on the left graph. However, what we need is the inverse function, that is, the function that gives the humidity level. This is shown in Fig. 5.5 on the right graph. To find the expression for the function, we assumed that the arrangement of the function's points was similar to a arctangent. From this assumption, we found the following expression:

$$Voltage = 2.041 - 0.994 \cdot arctan(0.119 \cdot Humidity - 4.515) \tag{5.2}$$

This curve is drawn in orange on the left graph. It seems to match the points correctly, but for this we decided not to use the data at the 100% rate because it seemed not to follow the trend of the other data. This is not a big deal because when using the station the plant will never be submerged only in water. From this equation, we can find the inverse function f^{-1}:

$$Humidity = \frac{1}{0.119}\left(tan(\frac{2.041 - Voltage}{0.994}) + 4.515\right) \tag{5.3}$$

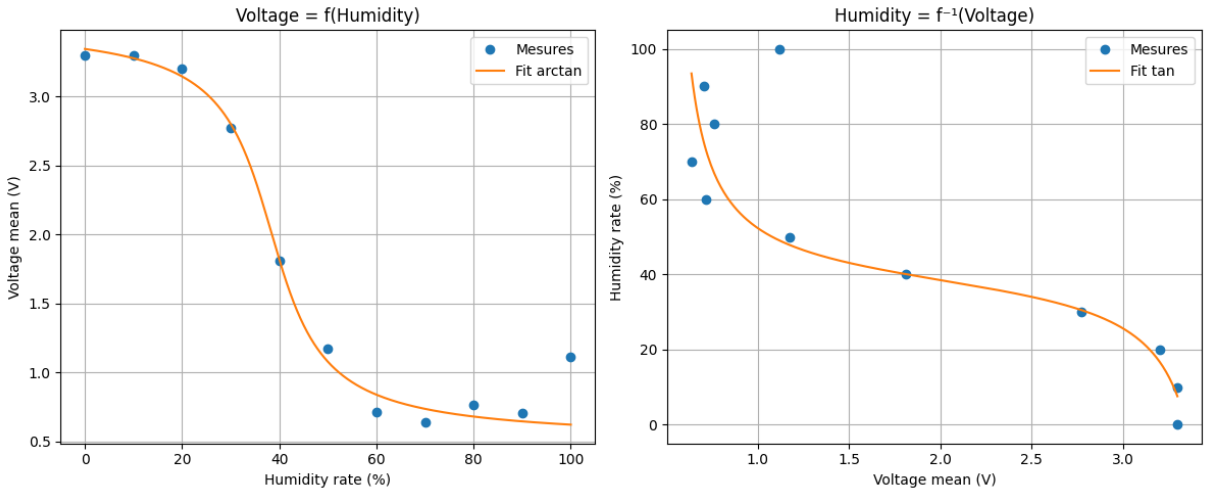


Figure 5.5: Transfer function (left) and inverse transfer function (right) deduced from the experiments

Noise

To better study the sensor, ensure its reliability, and understand the data used for the transfer function, it is necessary to look at the noise. To do this, we will conduct an other experiment. In fact this is exactly the same as for the transfer function, however, here we will focus on the variation of the measurements taken for each rate. This gives the graphs in Fig. 5.6.

The graph on the left is a box plot, while the one on the right shows the standard deviation for each rate. From these graphs, we can deduce that the higher the humidity rate, the more variable the measurements are. This is especially true for rates of 80, 90,

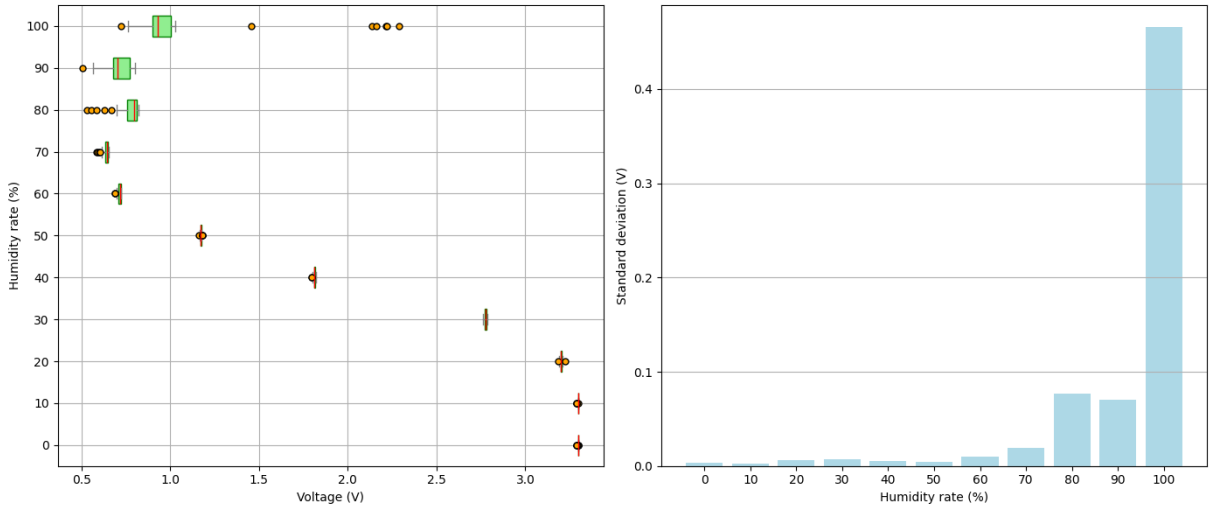


Figure 5.6: Box plot of measurements taken during the experiment (left), standard deviation for each humidity rate (right)

and 100%. If we look at the box plot, we can see that ambiguity in the measurements may appear, i.e., for example, a rate of 90% may give a voltage of 0.7V, which will then be interpreted as a humidity level of 70%. One might think that this is problematic, however, in practice, a humidity level higher than 70% is quite significant and will not normally occur. If we look at the standard deviation at the 70% rate, we see that it is approximately equal to 0.02 V, which corresponds to a precision of about 2.5% relative humidity. This level of precision is adequate for the measurements we aim to perform.

5.1.5 Validation and testing

To validate the humidity sensor, we first performed the experiment described in section 5.1.4 a second time. However, here we compared the humidity measured using the transfer function found earlier with the theoretical humidity. This gives the graph Fig. 5.7. We can see that two major measurement problems appear. When we are at 10% and 100% humidity. These two problems are already known and are due to sensor saturation and the transfer function, which does not take into account the value at 100% humidity. These two deviations should not, however, be a problem because we believe, and hope, that the plant will never be drowned or dried out. The other values ultimately seem to follow the correct values, even if the high values deviate more from the target value. This problem was also discussed in the section on noise.

In order to visualize the sensor's behavior in more real-world conditions, we conducted a second experiment. This involved watering a 50g pot of dry soil with 20ml every hour. This gives the graph Fig. 5.8, on which we can compare the measurements with what we should theoretically observe. We can immediately see that the graphs do not match at all. We believe this is due to the fact that the added water is not distributed evenly throughout the soil. In fact, in previous experiments, the soil was mixed to ensure uniform moisture. This is no longer the case here. So, at the beginning of the experiment, we had

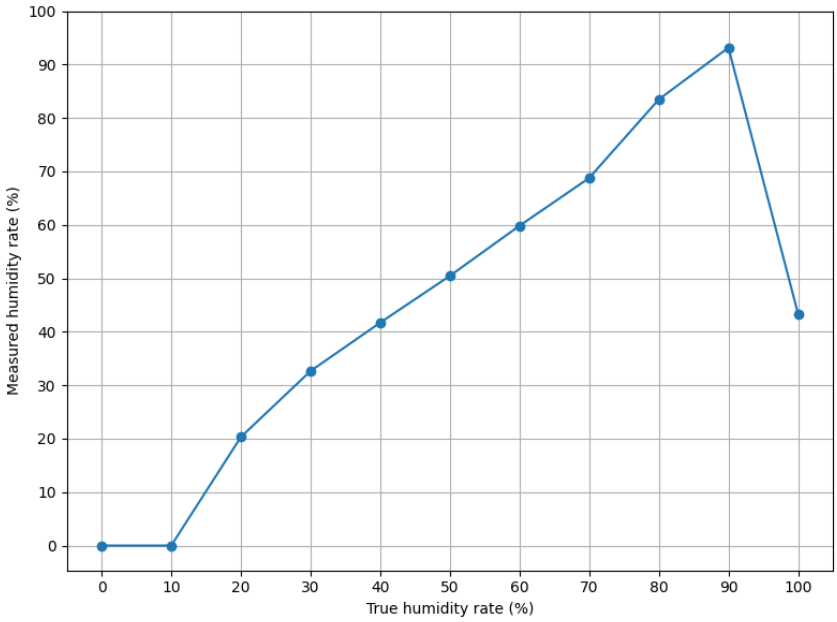


Figure 5.7: Function of the humidity estimated by the sensor compared to the humidity theoretically contained in the soil

water clumping above the dry soil and struggling to penetrate it, but still making contact with the top of the sensor. This resulted in a rapid increase in the moisture measurement. From the second watering onwards, the water started to trickle down to the bottom of the container, which had the effect of decreasing the measured humidity. When we stopped the experiment, we noticed that the sensor was actually partly surrounded by dry soil, while other areas of the pot had become almost entirely water.

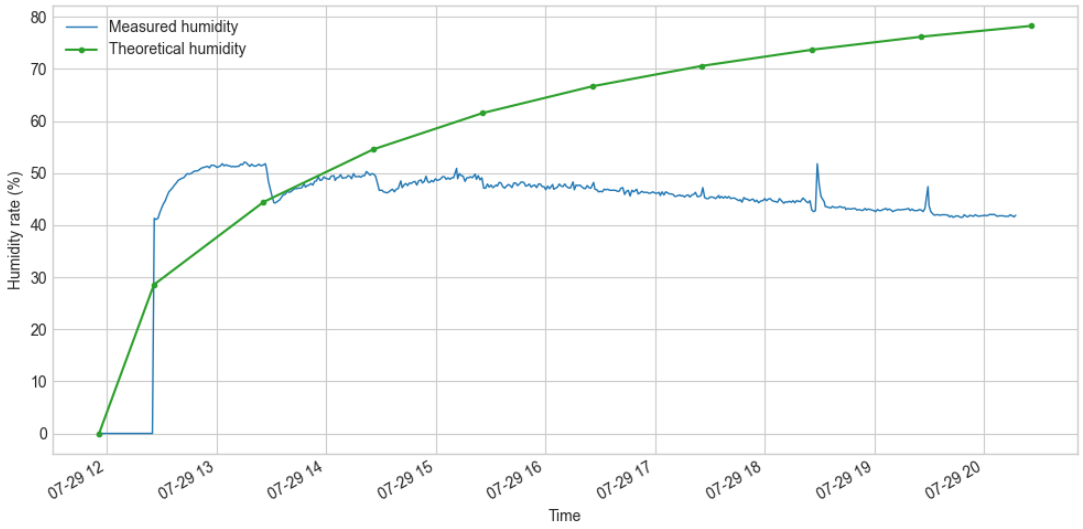


Figure 5.8: Comparison between the theoretical and measured variations in humidity by adding water every hour

Finally, this experiment shows that, under more realistic conditions where the plant is watered from above, the sensor has a major flaw: it can measure humidity only very locally.

This brings us to a final experiment. We took a container and refilled it with 100g of dry soil. Then, we added the necessary amount of water to obtain 30% humidity. Finally, we waited to see if the measured value stabilized after a long time. To prevent evaporation, a lid was placed on the container. We continued this experiment by adding enough water to reach a rate of 60%. The results are visible in Fig. 5.9. As expected, by allowing more time for the water to distribute itself in the soil, we obtain better results. 9 hours after the first addition of water, the measurement begins to stabilize around 34% humidity. This measurement isn't exactly what we were looking for, but the difference is still satisfactory. During the second addition, the humidity level stabilized much more quickly, about 2.5 to 3 hours. We will see in the last part of this chapter that this can be even a little faster and that, despite its slow response, we can still extract information from it. We believe that the speed of stabilization is influenced by the fact that wetter soil is less impermeable than dry soil, which makes it easier for water to diffuse into the container. However, here again, we have a difference in values. Where we were looking for 60% humidity, we find values between 55 and 57%.

In conclusion, although this newly added sensor is not completely accurate and is not very responsive, it can still provide us with new information keeping these two points in mind. It should be used in conjunction with the weight measurement which gives us another indirect indication of the overall humidity. Finally, it is important to note that all the measurements and experiments in this section were carried out with potting soil from the UCL greenhouses which has a different density than conventional soil, for example. If a conventional soil were to be used, then a transfer function other than equation 5.3 would have to be used.

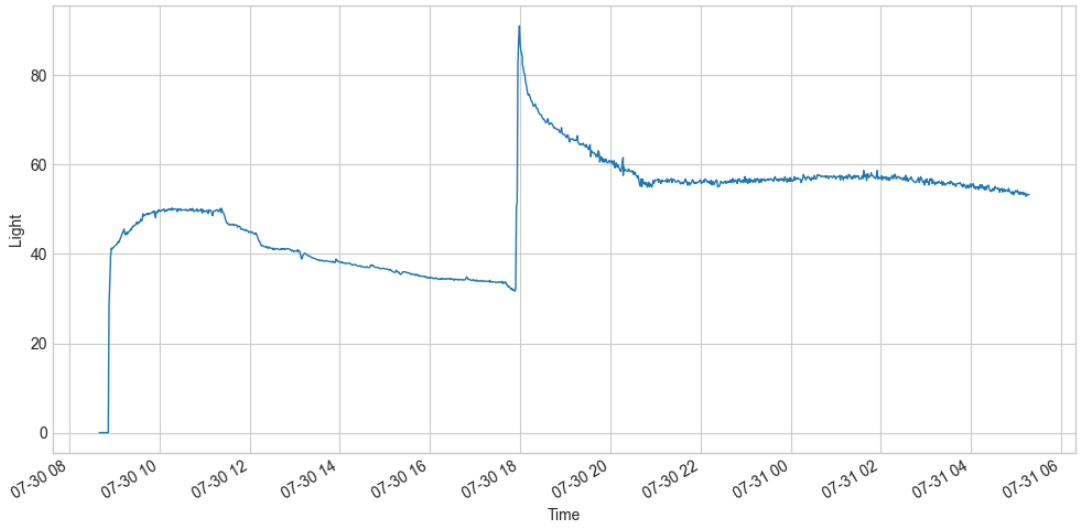


Figure 5.9: Evolution of humidity over a long period

5.1.6 post-processing

Except when the soil is watered, the moisture measurement should not vary greatly from one minute to the next. Therefore, as with growth, we add an average filter to our code. This averages over the last 30 minutes unless the increase is greater than 5% (twice the error calculated earlier). This way, the curve is smoothed but retains responsiveness during watering. The difference between the raw data and the smoothed curve is shown in Fig. 5.10.

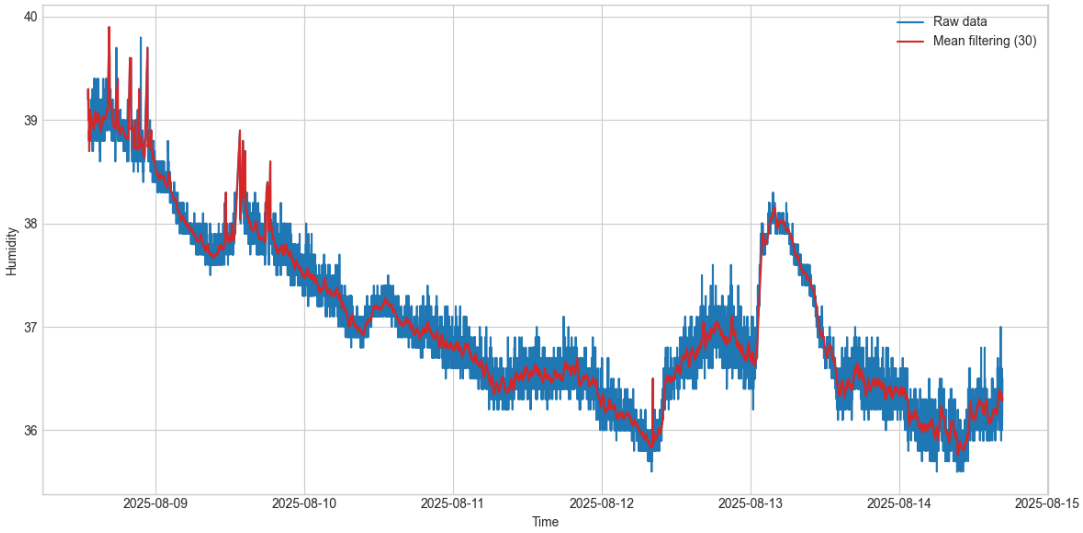


Figure 5.10: Comparison between raw data and filtering by average

5.1.7 Long-term measurement

As explained in the 3.3.6 section, we carried out a measurement over approximately 1 week. The humidity measurement is shown in Fig. 5.11. This shows that the humidity sensor is functioning correctly overall. We also note that it correctly recorded the two-stage watering performed on the fourth day (August 3rd).

Another observation can be made: occasional spikes appear in the measurements. These occur each time the station is restarted due to a camera error. We believe this phenomenon is related to the power supply because the sensor, which measures resistance at a reference voltage of 3.3 V, can return an abnormally high value if this voltage fluctuates when the power source is disconnected and reconnected. Due to a cascading effect, the reading is momentarily distorted before stabilizing to its true value. We consider this phenomenon to be normal, but note that it should simply be kept in mind when interpreting the data.

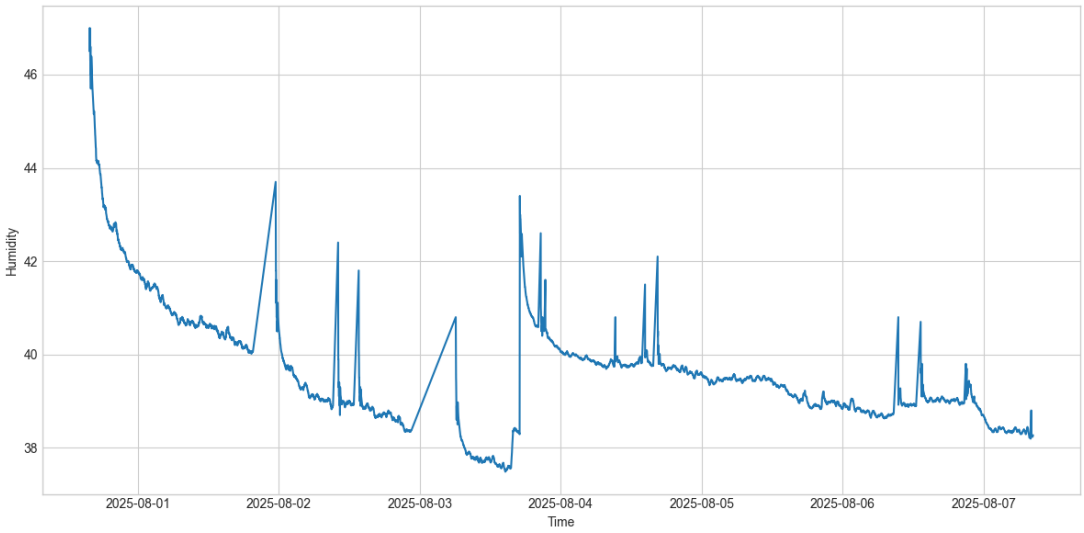


Figure 5.11: Humidity measurement during the long period

5.2 Light sensor

5.2.1 Interest of the addition

The integration of a light sensor would provide essential information on the light environment, a determining factor in plant growth and transpiration. Light directly influences stomatal opening and therefore transpiration, which impacts water loss measurements. By measuring light intensity, it becomes possible to explain certain daily or sudden variations in the data, including transpiration peaks or slowdowns in growth. This sensor would also make it possible to monitor the effects of a natural or artificial photoperiod on plant development, providing valuable context that can be correlated with data from the camera or scale, thus allowing for better analysis.

5.2.2 Overview

Like the humidity sensor, the light sensor is based on the voltage divider principle. There are modules similar to the humidity sensor[3]. However, the simplicity of the circuit allows us to easily make it ourselves at a very low cost. In fact, by adding up the costs of each component, we arrive at around 0.80€ for a sensor[4,5,6]. The welded result is shown in Fig. 5.12a.

To choose the resistance value, we started by measuring the photoresistor both in complete darkness and exposed to bright light. The results are as follows:

- Complete darkness: $\approx 35\Omega$

[3] https://www.amazon.nl/dp/B07DJ4WQV5
[4] https://www.amazon.fr/dp/B089YNCYG4
[5] https://www.amazon.fr/dp/B0CL6MDSHV
[6] https://www.amazon.fr/dp/B06VTTF5N9

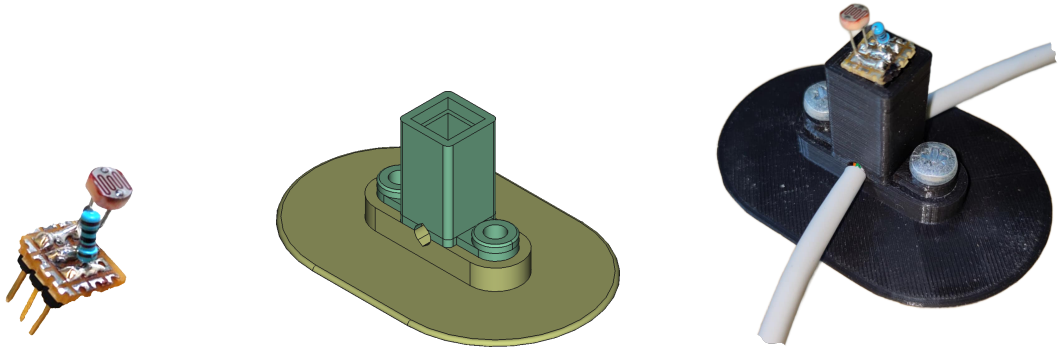
- Bright light: $+2000k\Omega$

From these measurements and the voltage divider formula 5.1, we can establish the table 5.2 which can help us in the choice of the resistor. We then find that the $10k\Omega$ resistor is the one which will give the greatest variation in voltage and therefore a better measurement of the photoresistance.

R	$V_{out,max}$	$V_{out,min}$	ΔV
100	2.4444	0.0002	2.4443
330	2.9836	0.0005	2.9830
1k	3.1884	0.0016	3.1868
10k	3.2885	0.0164	3.2721
33k	3.2965	0.0536	3.2429
100k	3.2988	0.1571	3.1417

Table 5.2: Variation of the output voltage depending on the chosen resistance

To avoid leaving the sensor connections exposed, a 3D printed support was designed. This can be seen in Fig. 5.12b. We chose to place this sensor between the station and the scale. In this way, and thanks to the support, the sensor will be, on the one hand, close enough to the plant to capture approximately the same light as it and at the same time far enough away not to receive any shadow produced by the plant, but also oriented upwards.



(a) (b) (c)

Figure 5.12: (a) Light sensor, (b) 3D support model, (c) Support an sensor

5.2.3 Integration into the station

In exactly the same way as humdity sensor, we will show the steps necessary to insert the light sensor into the station.

Branchements

The light sensor works exactly the same way as the humidity sensor, as they are both voltage dividers. We can then connect its output directly to the MCP3008 ADC we

installed earlier. The only thing to change is the channel on which we want to read the voltage, which must be different from the one for humidity.

As mentioned earlier, the sensor is located between the station and the plant. This placement takes advantage of the cable already connecting the scale to the station. Since the scale is already powered by 3.3V, we can connect directly to the same wires. The analog voltage uses a separate wire, but is still in the same cable. This cable then passes through the bracket through the central hole seen in Fig. 5.12c. Again, all connections are shown in a connection diagram in Appendix A.

Software adaptation

From a software perspective, adaptations have also been made. These changes are quite similar to those made for the humidity sensor. We can also reuse the MCP3008 instance created for it with the $mcp.read\_adc()$ function, which we use this time with a new, specially created attribute, $self.LIGHT$, corresponding to the light sensor's connection channel on the ADC.

To inform the user, light has been added to the measurement menu (Fig. 5.4). This value is given in Lux by f, which will be calculated in the next section:

$$Light\ in\ Lux = f(analog\ voltage)$$

As with the humidity sensor, the pipeline is also updated to display the measured brightness, Fig. 5.13.

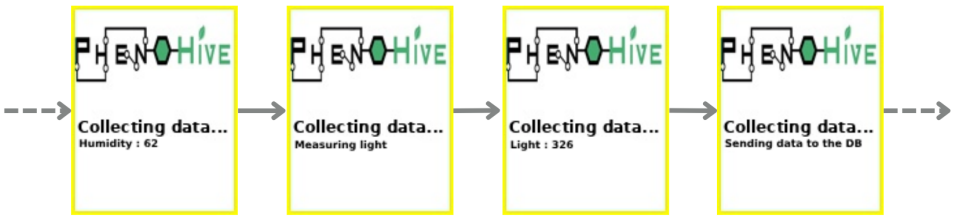


Figure 5.13: New display including light in the measurement pipeline

Finally, the light was added to the sending in the database and in the display on Grafana.

5.2.4 Sensor characterization

Transfer function

As with our humidity sensor, we need to define a transfer function to convert the measured voltage to the light measurement unit. The problem is that several measurement units exist for this one, but they don't all mean the same thing. Among them, we have three main ones [34, 35, 36]:

- The Candela (cd): is the unit of measurement of luminous intensity in one direction.

- Lumen (lm): is the unit of measurement for luminous flux. One lumen is equivalent to one candela steradian, which is therefore light passing through a surface.
- Lux: is the unit of luminous illuminance, that is to say the energy produced by 1 lumen per $1m^2$.

In our case, we will mainly use lux because it's more convenient and is generally used for photoresistors. However, we will also use lumens, which are used as a unit for light bulbs.

Now that we have established the basics of units, we can set up our experimental protocol:

1. First, we select different bulbs with different lumens. The different values are listed in the table Tab. 5.3.
2. Next, we take measurements for each bulb at different distances. The different measured distances are shown in the table below Tab. 5.3. A measurement in complete darkness was also taken.

Bulb type	Lumens	Distance [cm]									
Classic light bulb	806	5	8	14	16	19	35	39	42	69	99
G10 type LED bulb	345			14				39		69	99
G10 type LED bulb	230			14				39		69	99
G10 type LED bulb	200			14				39		69	99
Complete darkness	0						/				

Table 5.3: Parameters used to define the light sensor transfer function

3. For each bulb and each distance, we run a Python program similar to the one used for soil moisture. We then collect the tensions for all cases. For each measurement, around a hundred measurements were taken at intervals of 0.5 seconds.

From this data, we can attempt to find the transfer function linking voltage to lux. To do this, we have four choices: use a lux meter, use a smartphone with a lux measurement app using the built-in photoresistor, use a datasheet, or use a formula linking lumens and lux. None of these solutions are perfect.

The first solution, the lux meter, obviously seems the best, this device is accurate and is directly suited to our needs. Its main drawback, however, is that it is somewhat expensive, and it was therefore not possible to obtain one for testing. We therefore did not choose this solution.

The second solution, using a smartphone, also seems easy to use and less expensive than the previous solution. Despite this, we noticed that the values given were very different from those given by the two solutions that follow.

The third is the use of a datasheet. Here again, several problems oppose us. First, most of the photoresistors found did not have a code to identify them, this makes it difficult to find a datasheet. This photoresistor is however a classic photoresistor and therefore more than likely from the GL55 range or a GL5528 which are very widespread (especially in the kits from which this one comes) [37, 38, 39]. Finally a datasheet was found in which we can retrieve the following graph Fig. 5.14.

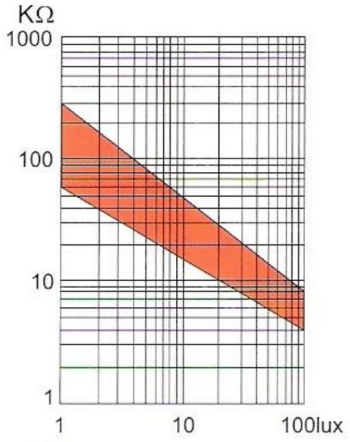


Figure 5.14: Photoresistor Datasheet [9]

The second problem is that this graph does not give a precise straight line, this is due to the fact that device-to-device variations can appear during the manufacturing of components. Despite this problem, we can still derive a function from this graph. To do this, we take two points from the orange area. Here we will take (1, 90k) and (100, 7k). Knowing that the function is a straight line in the logarithmic domain, we know the form of the function which is the following:

$$log(R) = a \cdot log_{10}(L) + b$$

Finally we find a = -0.555 and b = 4.954 or:

$$log(R) = 4.954 - 0.555 \cdot log_{10}(L)$$

Knowing the relationship between voltage and resistance of a voltage divider:

$$V_{out} = 3.3V \frac{R}{10k + R} \iff R = \frac{10k}{\frac{V_{in}}{V_{out}} - 1}$$

Which finally gives our first transfer function as well as its inverse:

$$L = 10^{\frac{4.954 - log(\frac{10k}{\frac{3.3}{V_{out}} - 1})}{0.555}} \iff V_{out} = \frac{3.3}{1 + \frac{10k}{10^{4.954 - 0.555 log(L)}}} \tag{5.4}$$

Finally, the last way is to find a relation between lumens and distance which are known values and Lux. The most commonly used formula is [40]:

$$Lux = \frac{lumens}{4\pi \cdot distance^2}$$

This formula can be understood as the dispersion of light on a sphere with a radius equal to the measuring distance. However, this formula does not take into account the difference between the bulbs used. In fact, the classic bulb has a radiation on a wider beam angle than the G10 type bulbs. To take this into account, we then use another formula including this beam angle:

$$Lux = \frac{lumens}{\pi \cdot distance^2 \cdot tan(\frac{\theta}{2})^2}$$

The denominator then still corresponds to a surface but this time depends on θ the beam angle of the bulbs. These angles were estimated for each bulb by placing a large piece of cardboard directly behind it and then measuring the angles formed by the light. These estimates are shown in Table 5.4. Comparing them with the ranges given by [41], we see that we are well within the range for each type of bulb.

806lm	345lm	230lm	200lm
230°	80°	70°	80°

Table 5.4: Estimate each beam angle per bulb

Finally with the data from the experiment and the angles, we can plot a graph of voltage versus Lux Fig. 5.15.

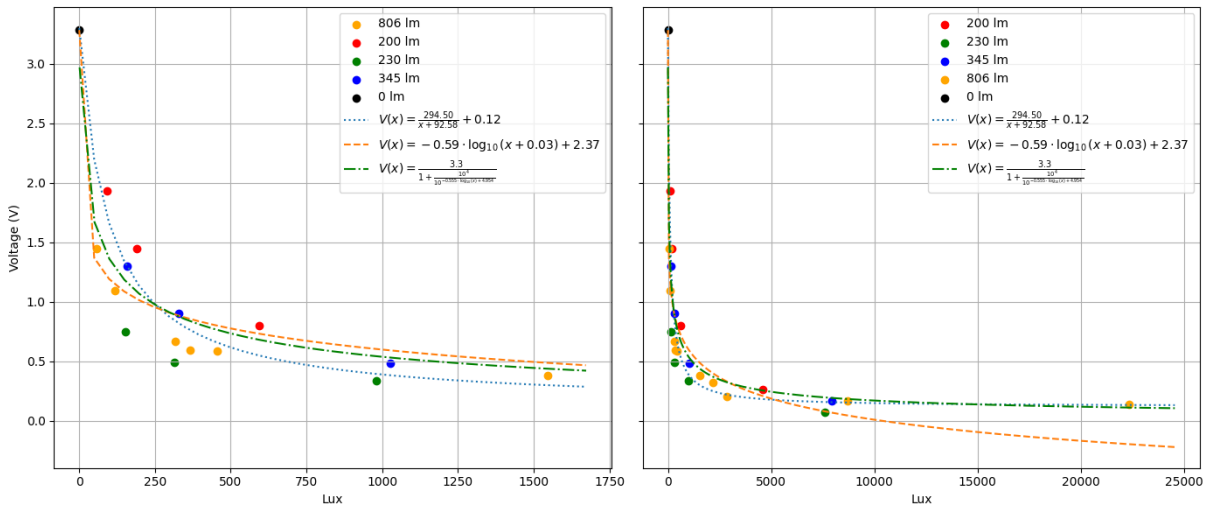


Figure 5.15: Comparison of different potential transfer functions

In this graph, each point represents the average of the measurements made for a bulb and a distance. Each color designates a different bulb. A first thing we can observe is that the sensor, when in the dark, saturates at a voltage of 3.3V which is normal and desired. Secondly, we can see that the points of the same color (corresponding to the same bulb) cluster together in distinct bands on the graph, but these bands are not arranged

according to the nominal luminous flux (in lumens). This is probably due to the fact that the beam angles were estimated because it was complicated to measure them precisely. We can finally observe that the trend of the points is logarithmic or hyperbolic. For this we try to fit two generic functions:

$$\frac{a}{x+b}+c \qquad\qquad a \cdot log_{10}(x+b)+c$$

This gives us the function in blue and yellow on the graph. We also display the function found thanks to the datasheet, equation 5.4. We can thus compare these three functions in order to choose one. First, we observe that the logarithmic function did not correctly fit the high light points, so it is discarded. Between the two remaining functions, the similarity is great. However, we note that the hyperbolic curve corresponds better to low-intensity points. Another argument is that the curve found by the datasheet does not reach 3.3V when the lumens are at 0. This is due to the logarithm used. We therefore keep as transfer function as well as its inverse:

$$V = \frac{294.5}{Lux+92.58}+0.12 \iff Lux = \frac{294.5}{V-0.12}-92.58$$

Noise

As with the humidity sensor, we can also analyze the variations in the measurements. This gives us the graphs Fig. 5.16. These two graphs show us that the measurements are very stable. The maximum standard deviation is at 0.04 V where we had 0.4 V for the humidity sensor. In conclusion, this means that the sensor is quite accurate. The biggest error is then due to the approximation of the transfer function which is based on measurements at different lumens and whose beam angles have been approximated. That said, the purpose of the measurement made by this sensor is to have a rather relative idea of the illumination as a function of the day and not a precise measurement in Lux.

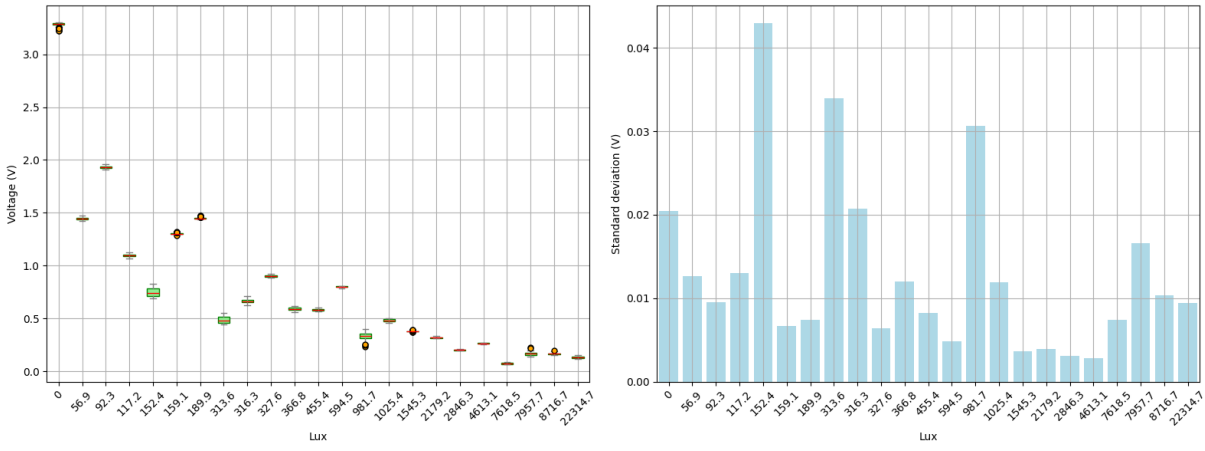


Figure 5.16: Box plot of measurements taken during the experiment (left), standard deviation for several light intensities (right)

5.2.5 Validation and testing

To validate the light sensor, we simply let the device run for 48 hours. This gives us the graph shown in Fig. 5.17. In it, we can observe several things. First, we can very clearly see the two day/night cycles. Indeed, when we measure at night, we find values between 1 and 5 lumens, which corresponds almost to complete darkness. At first glance, daytime measurements might seem very noisy. However, both days on which the measurements were taken were cloudy. Since lumen measurements can vary very quickly due to their inherent scale, it is normal to see such significant differences from one minute to the next. One last thing we can see is that the period of highest intensity is between 9:00 a.m. and 11:00 a.m. This corresponds to the hours when the sensor could receive direct sunlight at its location relative to the window that let in the light.

The sensor therefore seems to give a fairly good idea of the light that would be received by the corn plant depending on the hours of a day and depending on the different days.

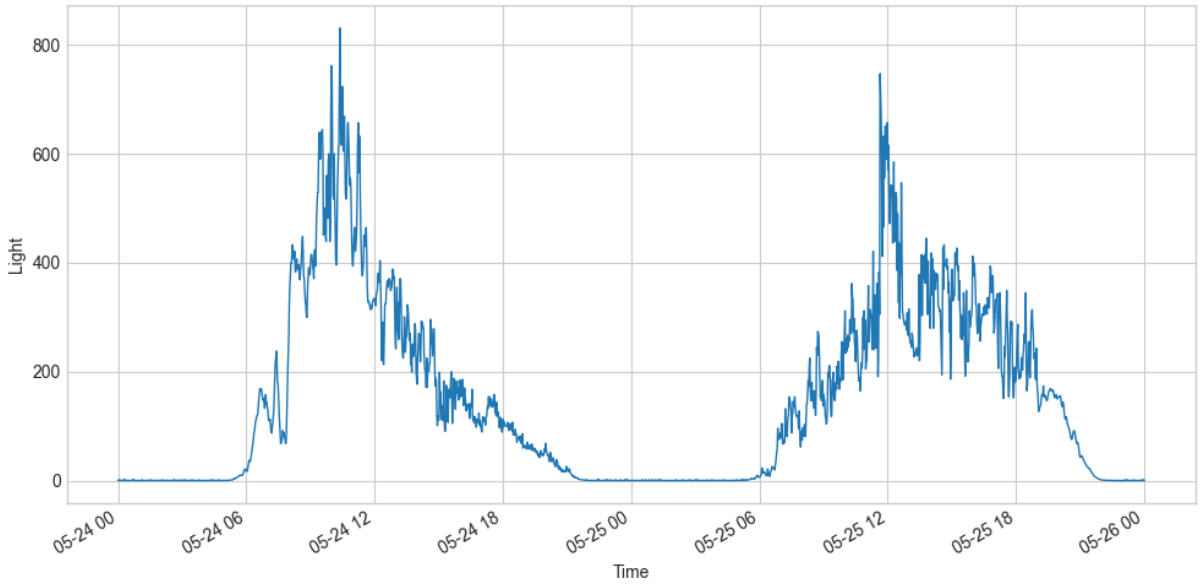


Figure 5.17: Light intensity measurement over 48 hours, on cloudy days

5.2.6 Long-term measurement

During the long measurement period, we were lucky enough to have two consecutive days with almost no clouds. The results are shown in Fig. 5.18. This proves that the sensor can be both very responsive to changing light, as in Fig. 5.17, but also quite accurate and low-noise. We therefore do not need an average or median as with other measurements.

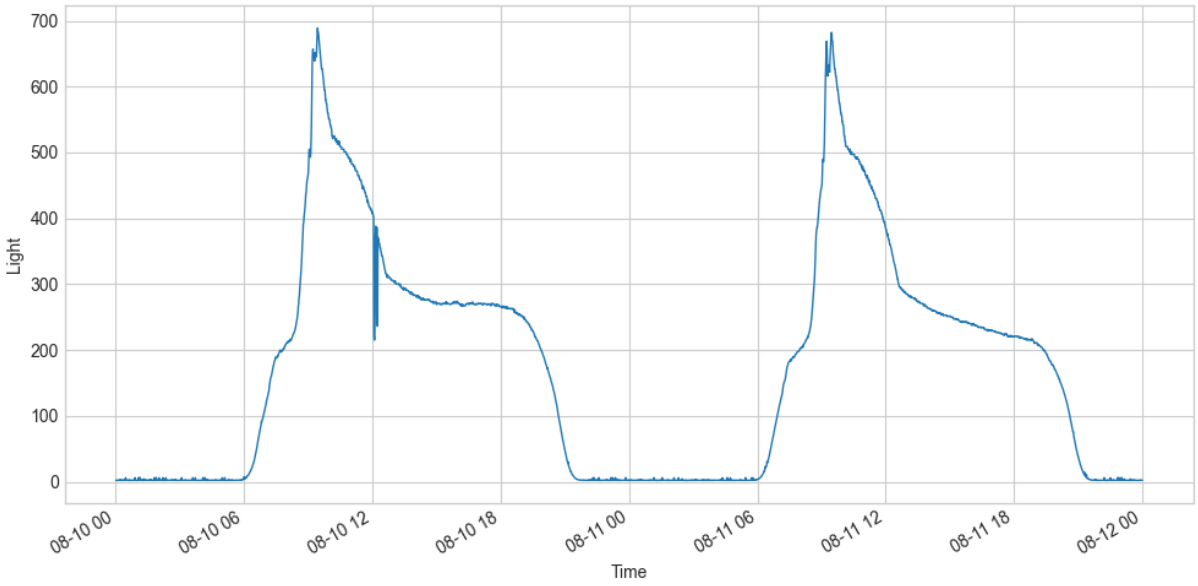


Figure 5.18: Light intensity measurement over 48 hours, on cloudless days

5.3 Use and link between measures

From the new measurements, we can deduce new information and make new connections.

5.3.1 The different water exchanges

This part concerns the exchange of water between the earth, the plant and the air.

Determining Initial Masses

We initially want to know the mass of dry soil used. To do this, we have two options:

- The first is to weigh the soil when it is very dry (for example, by drying it in an oven).
- The second is to deduce this mass from the mass of the pot M_{pot} and the soil moisture content measured as a fraction:

  $$H(t) = \frac{M_{\text{water,soil}}(t)}{M_{\text{soil,dry}} + M_{\text{water,soil}}(t)},$$

  The total mass measured at time t is

  $$M_{\text{tot}}(t) = M_{\text{pot}} + M_{\text{soil,dry}} + M_{\text{water,soil}}(t)$$

  At the initial time, we obtain:

  $$M_{\text{soil,humid}}(t_0) = M_{\text{tot}}(t_0) - M_{\text{pot}}.$$

  The humidity $H(t_0)$ then allows us to find the dry mass of the soil:

  $$M_{\text{soil,dry}} = [1 - H(t_0)]\ M_{\text{soil,humid}}(t_0).$$

Mass Balance During the Experiment

At any time t, we calculate:

$$M_{\text{water,soil}}(t) = \frac{H(t)}{1 - H(t)} M_{\text{soil,dry}},$$

$$M_{\text{water,total}}(t) = M_{\text{tot}}(t) - M_{\text{pot}} - M_{\text{soil,dry}},$$
$$M_{\text{water,plant}}(t) = M_{\text{water,total}}(t) - M_{\text{water,soil}}(t).$$

Interpretation of Variations

The differential water balance of the system is:

$$\frac{dM_{\text{water,soil}}}{dt} = -E - A,$$

$$\frac{dM_{\text{water,plant}}}{dt} = A - T,$$
$$\frac{dM_{\text{water,tot}}}{dt} = -E - T,$$

where:

- E: evaporation flux (soil $\rightarrow$ air);
- T: transpiration flux (plant $\rightarrow$ air);
- A: water absorption flux by the plant (soil $\rightarrow$ plant).

Link with growth

Using this information, students will be able to relate water exchange to plant growth. They will be able to determine whether a plant grows or shrinks based on transpiration or the water it contains.

5.3.2 Link with luminosity

No directly calculable information can be deduced from the measurement of light alone. However, the student can relate it to water exchanges to determine whether the plant retains more or less water or whether it transpires more when the light varies. It will also be possible to assess whether light has a direct impact on growth.

Chapter 6

Online database and data management

This chapter will be divided into two parts. First, we will see how to ensure that the workstation and the database no longer need to work on the same local network. In the second part, we will set up a system for backing up data to the database.

6.1 Online database

In this section, we will first explain why we want to export the database online, then we will present the different elements of network connectivity, apply this theory to our project by presenting the Ngrok tool used subsequently to put our server online and finally we will explain the concrete steps of setting up this tool and the modifications that go with it.

6.1.1 Motivation

The idea of the basic project is that each student (or group of students) has a station and a plant of their own. They could then use the station from home without having to go to the greenhouses, which, as explained in section 2.1, are difficult to access for so many students at once.

The problem with the current prototype is that the stations and the database must be connected to a local network to be able to communicate via the ports. The stations must then remain in the greenhouses or elsewhere, but all in the same location and close to the module hosting the database, in this case, a Raspberry Pi 3. We are therefore missing the initial objective for the moment.

The goal of this part is to make the database accessible over the internet. This way, we can send data to it from anywhere as long as the station is connected to Wi-Fi.

6.1.2 How network connectivity works

Network connectivity defines the ability of a device (computer, server, etc.) to exchange information with other devices via a network. This network can be local (LAN, Local Area Network), such as in a home or business, or global, such as the Internet.

For this communication to work, several elements come into play:

- **Transmission Medium**: Data travels via a physical medium (Ethernet cable, fiber optics) or wirelessly (Wi-Fi, 4G/5G). In our case, the station and server use Wi-Fi.
- **IP Address**: Each connected device has an IP (Internet Protocol) address, a unique number that identifies it on the network. It's a bit like a postal address that lets data know where to go. There are IPv4 addresses (e.g., 192.168.1.10) and IPv6 addresses (longer, to accommodate more devices).
- **Domain Name and DNS**: A domain name (e.g., example.com) is easier to use than a raw IP address. The DNS (Domain Name System) is a service that translates a domain name into an IP address.
- **Ports**: A port is a number indicating the target service or application on a device. For example, InfluxDB uses port 8086 by default, and Grafana uses port 3000.
- **Protocols**: Protocols are communication rules. The IP protocol manages the addressing and routing of data packets. Other protocols exist, such as HTTP, which allows browsers and web servers to exchange pages and information. HTTPS is the secure version of HTTP: it encrypts the data exchanged so that no one else can read or modify it during transfer.
- **Data Routing**: When a device sends data, it is broken into packets. Each packet contains the destination IP address, the port, and the payload (the data). These packets pass through various devices (routers, servers, relays) before reaching their destination.

Thanks to all these elements, network connectivity allows us to browse the Internet, exchange files, and communicate in real time even if we are not on the same local network.

6.1.3 Connectivity in our project

Now that the connectivity foundations have been laid, we will explain the different elements and connections we will implement for our project. To better understand the different elements, a diagram is shown in Fig. 6.1.

Here is each element and how they interact with each other:

- **Phenohive stations:** These are present at the beginning of the chain. Their purpose is to send data to the database. While until now they could send them directly locally with $http://localhost:8086$, we now want to add an intermediate gateway

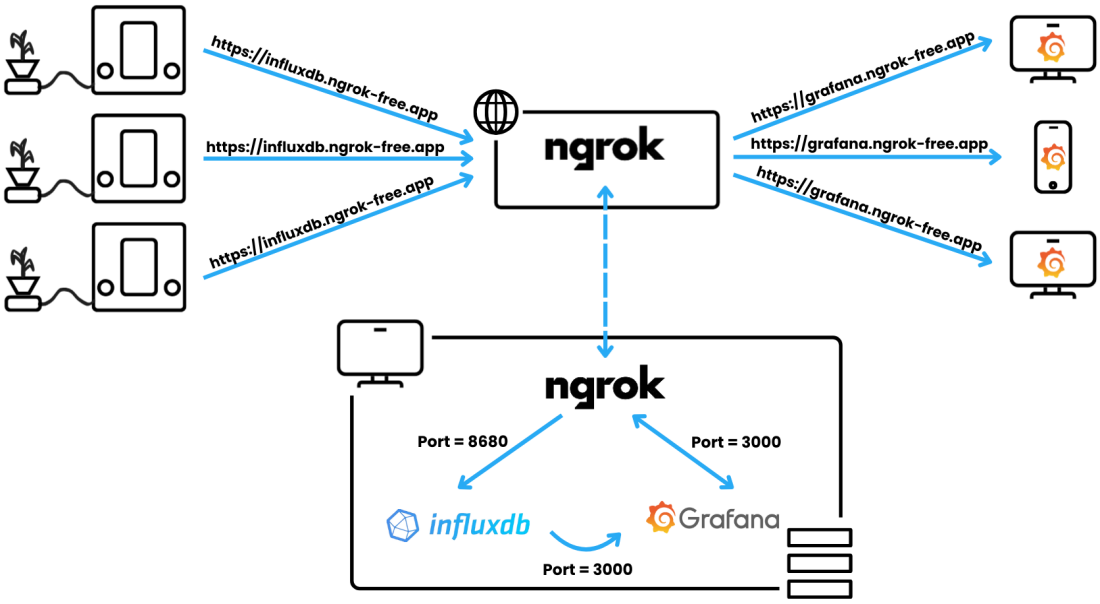


Figure 6.1: PhenoHive project network connectivity diagram

via the internet to allow the exchange of data at different locations. For this, we will use Ngrok, which we will present in a few moments. Before sending, the station must prepare packets. These contain the measurements, the station number that differentiates it from the others in the database, and finally, the IP address corresponding to the domain name that leads us to Ngrok. Although the domain name will no longer be local, it will still be stored in the station's *config.ini* file, as before.

- **Ngrok:** This is the central point of going online. Ngrok is a tool that allows you to expose a local server to the internet via a secure tunnel. It works by creating an outbound connection from your local machine to Ngrok's cloud service, which then forwards external requests to your local server. This way, the local machine is accessible from anywhere without opening ports or modifying network configurations. Ngrok generates a temporary public URL (HTTP or HTTPS) that redirects traffic to a local machine, which is very useful for testing or sharing a project without having to configure a server or modify our network. The main advantages of Ngrok are that it's free and easy to set up [42]. In our case, Ngrok therefore communicates on the internet via two domain names, one for influxDB and another for Grafana (https://influxdb.ngrok-free.app and https://grafana.ngrok-free.app in the figure). On our computer/server, it communicates locally with port 8086 for influxDB and 3000 for Grafana.

- **Server**: This is the element that hosts InfluxDB, Grafana, and Nrgok, as explained above. In our case, we will use a laptop running Windows 11, but in the previous thesis [2], a Raspberry Pi 3 was used. In the server, the data arrives on the local Ngrok and is then sent over port 8086 to InfluxDB, which in turn sends the data via port 3000 to Grafana (which allows data visualization).

- **Users:** These are represented by the computers and the smartphone in Fig. 6.1. In order for them to be able to visualize the measurements taken, they must be able to access Grafana. This is done, as explained earlier, via Ngrok once again. The advantage is that the user can access the measurement from anywhere as long as they are connected to the internet.

6.1.4 Steps for setting up remote database hosting

In order to set up access to the database via the Internet on our computer, we followed the steps that will be presented below.

Installing the database locally: InfluxDB and Grafana

To create the database, we decided to keep InfluxDB and Grafana, which remain perfectly suited to the PhenoHive project [1, 2]. Keeping the same database system also avoids rewriting most of the code and preserves all existing functionalities.

The installation procedures for both InfluxDB and Grafana are available in their official documentation:

- InfluxDB: `https://docs.influxdata.com/influxdb/v2/install/?t=Windows`
- Grafana: `https://grafana.com/grafana/download?platform=windows`

The configuration can then be carried out as previously implemented on the Raspberry Pi (see Appendix in [2]), but this time on the server host machine (in this case, a Windows computer).

To make the database startup automatic, we can create a service[1] to run a program in the background. For this, we need a new tool. We will use NSSM. NSSM (Non-Sucking Service Manager) is a Windows tool that allows you to launch any application or script as a Windows service, even if it was not designed for that purpose. Unlike the built-in Service Manager, NSSM cleanly manages application startup and shutdown, and offers options for redirecting output, automatically restarting on failure, and configuring environment variables [44].

Installation is via the following link: `https://nssm.cc/download`

Once installed, we can launch the executable file using a command prompt. To do this, we will use the command:

$$C : \backslash path\ to\ the\ file \backslash nssm.exe\ install\ InfluxDB$$

A window will then appear, Fig. 6.2, and this is where we can enter the parameters of our service. Using NSSM is simple, we just need to enter in the *Path* field, the path to the executable, here, influxdb.exe downloaded earlier, the *Startup directory* field is set

[1]Service, also called a daemon, allows programs to run from startup and in the background [43].

by default in the same folder as the executable file but it can be changed. The name can also be changed if necessary. Finally we can press *install service* to create it. Now that our service exists, we only have one step left which is to start it. To do this, we use the command:

$$C : \backslash path\ to\ the\ file \backslash nssm.exe\ start\ InfluxDB$$

From now on, the InfluxDB database will be started whenever the computer is turned on and we can access it with any browser with the url below because influxDB is installed by default on port 8086.

$$http : //localhost : 8086$$

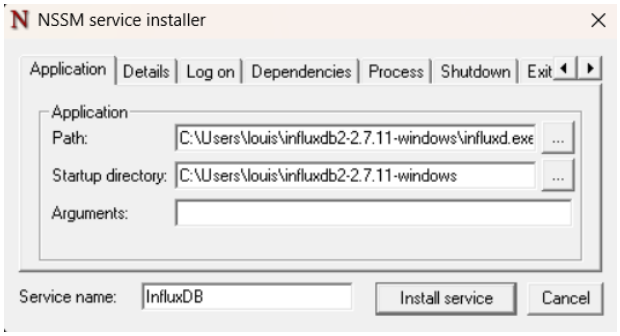


Figure 6.2: NSSM menu

For Grafana, we don't need to create a service. If the installation was done via the .msi file, the installer should have already created a service by itself. To check this, we can open the Service Manager using **Windows + R** and entering **service.msc**. From there, we can find Grafana in the list and at the same time, we can also check for InfluxDB.

Creation of a domain and putting the database online: NGROK

Now that our database is functional and easily accessible, we need to make it available online. To achieve this, we will use Ngrok.

The installation files are available at the following link: `https://ngrok.com/downloads/windows?tab=download`

Setting up Ngrok is done in several steps which are as follows:

- After creating an account, we obtain an **AUTHTOKEN** This allows us to link our machine and client to our Ngrok account. We will use it later.
- Basically, Ngrok creates non-fixed domains. This means that each time the tunnel is turned on, the domain name will change. This is impractical for our project, which relies on a fixed domain name to communicate with the database. One solution could be to retrieve the domain name directly from the workstation each time it changes, but the simplest solution is to create a fixed domain name. To do this, in Ngrok dashboard, we can go to the **Domain** section and click on **+ New Domain**. Thanks to this step, we can retrieve a URL like: **generated_name.ngrok-free.app**. This name will remain fixed and is the one we will use from now on.

- In order to define the parameters of our tunnel, we will create a YML file. A YML (or YAML) file is a text file used to represent data in a readable and structured way. It is used to define parameters or describe data in the form of keys and values, with indentation by spaces. Its syntax is simple and clear, which makes it easy to use. By default, Ngrok reads this YML file at the following path: $C:\backslash Users\backslash user\_name\backslash .ngrok2\backslash ngrok.yml$ but thanks to the steps that follow, we can choose the location and create this file in the same folder as the executable file ngrok.exe. The contents of this file are as follows:

```yaml
version: "3"
agent:
  authtoken: AUTHTOKEN

tunnels:
  influxdb:
    proto: http
    addr: 8086
    domain: generated_name.ngrok-free.app
```

Listing 6.1: Configuration file `ngrok.yml`

  In this file, we have different sections:

  - **version**: This section specifies the version of the configuration file format used, here version 3. It is only meant to indicate the version of the configuration file for Ngrok.
  - **agent**: The agent is the Ngrok program running locally on our machine and establishing tunnels to the outside. It has as the parameter **authtoken**, which is a unique key that allows the agent to authenticate with our Ngrok account.
  - **tunnels**: In this section, we can define different tunnels. This is where the tunnel named **influxdb** is defined, which redirects incoming **HTTP** traffic on the public domain **generated_name.ngrok-free.app** to port **8086** on the local machine, where our InfluxDB service is running. This tunnel thus allows access to this local service from the internet via a public URL provided by Ngrok. This connection is secure because Ngrok uses HTTPS. So we have an HTTPS connection to Ngrok and a local HTTP connection between Ngrok and InfluxDB.

- Our last step is to set up automatic startup via a service in the same way as for InfluxDB. We will proceed almost the same way with NSSM but we will add a step. To make the launch cleaner, we will create a .bat file before our service. A .bat file is a text file containing a series of commands that are automatically executed by the Windows command prompt. A .bat file is actually more or less similar to a .sh file on UNIX [45]. The bat file looks like this:

```
@echo off
cd /d "C:\path_to_directory\ngrok"
ngrok start --config "C:\path_to_directory\ngrok\ngrok.yml" --log=ngrok.log
    --log-format=logfmt influxdb
```

Listing 6.2: Script `.bat` to start Ngrok with Influxdb

In this file we have:

- **@echo off**: This allows you to not have a display in a command prompt.
- **cd**: Allows you to move to the correct directory.
- **ngrok start influxdb**: Allows you to start the Ngrok **influxdb** tunnel. The **config** option is used to specify that the YML file we created earlier should be used, while the **log** option specifies the creation of a log file, which can be useful for debugging.

This is the file that will be launched by the service. To create it, we will proceed in the same way as for influxDB.

$$C:\backslash path\ to\ the\ file\backslash nssm.exe\ install\ Ngrok$$

Then we fill in the paths to the ngrok.bat file.

$$C:\backslash path\ to\ the\ file\backslash nssm.exe\ start\ Ngrok$$

Grafana goes online for visualization

Now that our database is writable via the internet, we also need to enable online viewing on Grafana. To do this, we follow exactly the same steps as in the previous section where we connected InfluxDB using Ngrok. To avoid repetition, we will only mention the differences in the steps.

First, we need to create a new domain name on the Ngrok dashboard. The free version only allows one domain name per account, so you must either upgrade to the paid version or create a new account. Here, we have chosen to open a second free account.

We need to create a second YML file called *ngrok_grafana.yml* and a new .bat file called *ngrok_grafana.bat*. The first one then becomes:

```
version: "3"
agent:
  authtoken: SECOND AUTHTOKEN

tunnels:
  grafana:
    proto: http
    addr: 3000
    domain: second_generated_name.ngrok-free.app
```

Listing 6.3: Configuration file `ngrok_grafana.yml`

Where the AUTHTOKEN is changed to match the new account, the address is changed to match the Grafana port and the domain name is changed to match the newly generated name.

The ngrok_grafana.bat file is:

```
@echo off
cd /d "C:\path_to_directory\ngrok"
ngrok start --config "C:\path_to_directory\ngrok\ngrok_grafana.yml" --log=ngrok.
    log --log-format=logfmt grafana
```

Listing 6.4: Script `.bat` to start Ngrok with Influxdb

Once these two new files are created, we can launch the service in the same way as before. After these steps, we can view the data remotely.

Adapting the station code to the new database

To make the station compatible with the new database layout, we need to make some changes.

First, in config.ini, we need to change the URL to the new one provided by Ngrok. If the InfluxDB token has been changed, it should also be updated in this file.

Second, we need to add a line to the data we send to the database. It is as follows:

$$"ngrok - skip - browser - warning" : "true"$$

This line must be added because, when connecting to a URL generated by Ngrok, it displays a warning page by default, informing the user not to continue if they are not sure of the source of the link. By adding this line, the query bypasses this warning page to communicate directly with InfluxDB.

6.2 Backup system for the database

Until now, when the database was disconnected from the station, the data collected was not retrieved upon reconnection. However, each time a measurement was taken (whether the station was connected or not), the program stores the data in a CSV file. This means that the data is in memory and is not lost. However, until now, it was necessary to connect to the station via an SSH connection to manually retrieve the data, which was inconvenient. In this section, we will add a backup system to our machine to resend all data collected during a disconnection.

The new system is quite simple. We will add a new attribute to our station, *last_data_send_time*. This attribute allows us to know the precise date of the last data sent to the database. Using this, we now compare this date to the current time on each send and send all unsent data up to *last_data_send_time*. Then, once the send is complete, we can update the attribute to be equal to now. This way, all the "lost" datas are sent at once on the first connection. An example is shown in Fig. 6.3.

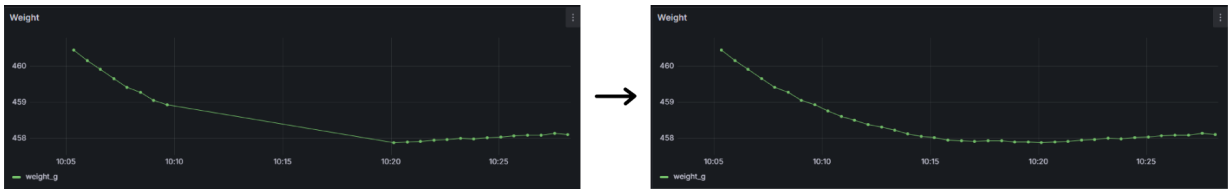


Figure 6.3: Difference between measurements without (left) and with (right) the data recovery system

There is, however, one problem that needs to be addressed, otherwise we wouldn't be able to achieve such a result. This is that the station and InfluxDB are not in the same time zone. Indeed, while our time zone (in Belgium) is UTC+2 (Coordinated Universal Time), InfluxDB's is based on UTC0. This poses a problem because, for example, if we send data with a time of 3 PM on the station, this means that for InfluxDB, it is 1 PM. For InfluxDB, we are therefore trying to send data in the future, which it does not accept at all.

A time correction is then made before sending the data. This correction is done automatically and takes into account the time zone in which the station is located. This means that the time will be corrected regardless of whether the station is located in the Americas, Asia, or Belgium.

Chapter 7

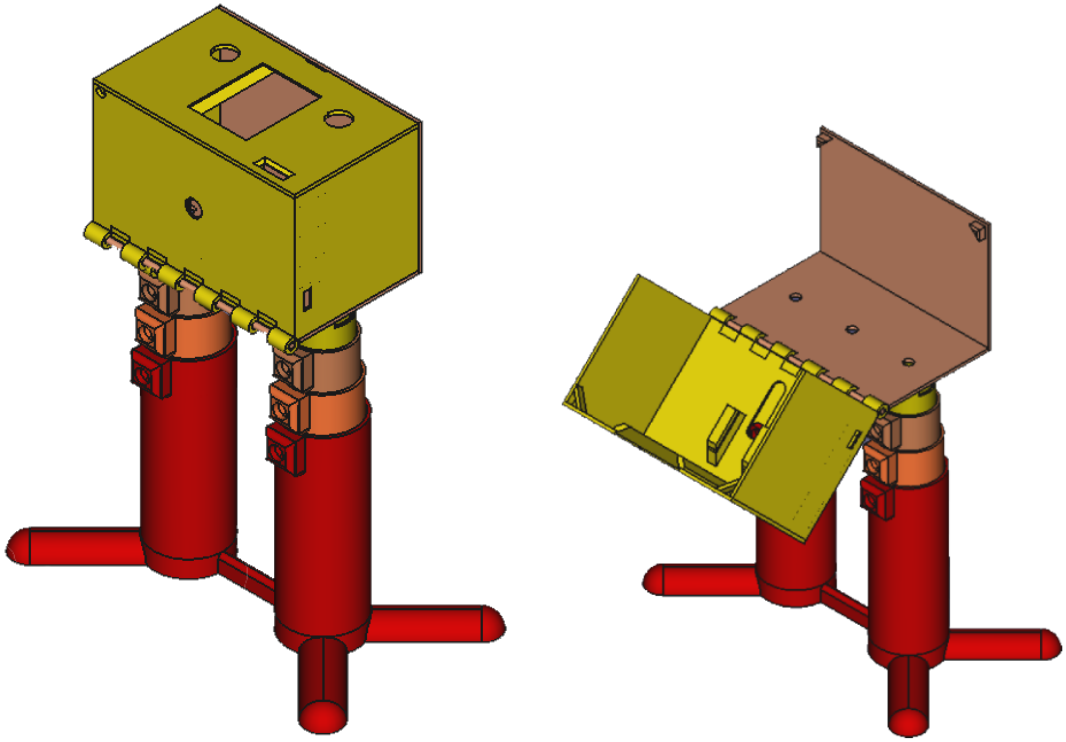
Improvement of the outer casing

Since the beginning of the project, the station's box has remained a prototype without ever being improved. It therefore has many flaws that make it impractical to use. This section explains how we went from prototype to the final, or almost final, box.

7.1 The main defects

As mentioned before, the box, which can be seen in Fig. 3.4, has several defects, here is a list:

- First, the box used to hold all the components is a box normally used by electricians. It is therefore a size of $20cm \times 12cm \times 7.5cm$ which is much too large to hold the few components of the station. This box is also drilled in an artisanal way by hand which gives an unclean and unfinished appearance to the station.
- The second flaw is more about maintenance than usage. The problem here is that elements inside the box are attached to the bottom of the box as well as to the lid. Since these elements are connected to each other by electrical wires, when you open the box you pull on them, which can damage or even disconnect them. In addition, they prevent us from fully opening the box.
- The third flaw is that the box has no feet. This is not very practical because, in its current state, the box cannot stand without a support placed underneath it. You therefore have to place something you have on hand, such as books, plastic cups, etc., on it. In addition, you have to position it at the right height so that the camera is aligned with the plant. This adds once again to the unfinished nature of the station.



(a) (b)

Figure 7.1: (a) Station outer casing closed, (b) Station outer casing open

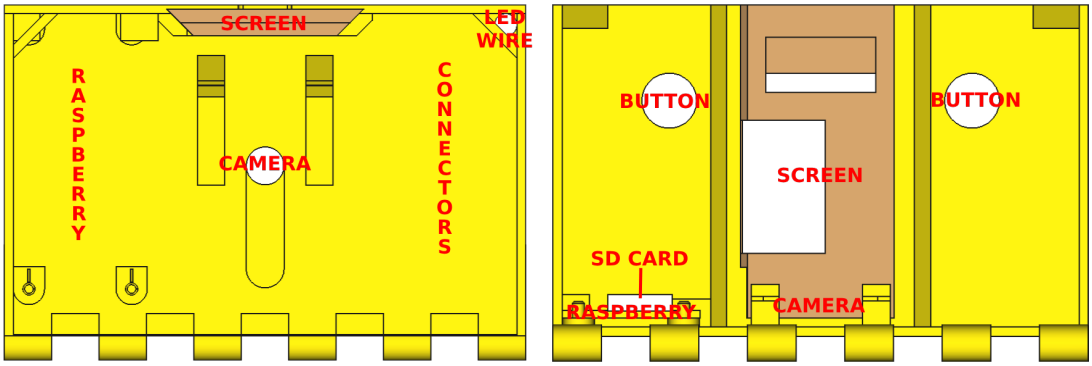


Figure 7.2: Location of the different elements of the station in the box; Left, face parallel to the plant, right, top face

7.1.1 Modifications and enhancements

To improve our box, we chose to use 3D printing, which gives us complete freedom regarding the shape and size. At the end of the section, we will make a price comparison with the old box. Finally, here is a list of all the elements that have been added or improved. To help you find your way around, the entire image of the box as well as the image of the location of each component are shown in the figures, Fig. 7.1 and Fig. 7.2.

- The size has been greatly reduced, $11.68cm \times 7.8cm \times 7.4cm$. Its height corresponds

to the length of the raspberry, the width corresponds to the length of the board supporting the screen and finally the length of the box allows to accommodate the raspberry the camera as well as the wire connectors and the relay. We can hardly reduce this size. This could be possible by placing the screen horizontally and more vertically but in addition to the main components mentioned above, we must not forget that we also have the wires which take up a lot of space.

- All the elements of the box are attached to the yellow part that you can see in Fig. 7.2. There is only one wire that does not pass through this part and that is the wire coming from the scale. This one passes through the second part of the box but does not interfere with the opening at all. To attach all the components, it was necessary to think of hooks that would allow us not to use screws. Indeed, the plastic used for printing is PLA which does not withstand the force and friction of screws well. So for the Raspberry we have a plastic interlocking in its holes initially intended for screws. The camera is simply slid into grooves. Finally, the screen is blocked between two plates, the orange part in Fig. 7.2. These two plates, once pressed and blocking the screen, are slid into the yellow part.

- A support was finally added under the box. This consists of 2×4 tubular parts that form a telescopic foot. The lowest part is the foot itself while the highest part, in contact with the box, is screwed to it. To prevent the pipes from falling back by themselves, small vertical square parts were added. They allow us to insert a bolt into which we will screw a screw so that it tightens the inner pipe. A similar principle is used to fix the box on the upper pipe. Finally, thanks to this system, the camera can be at a variable height between 21cm and 52cm. This is useful because the box may have to be moved during the same experiment when the plant becomes too large.

- The Raspberry Pi's useful ports have been added directly. Before this, it was necessary to open the box by unscrewing each screw to insert/replace the SD card or the power cable. With the new box design, the SD card can easily be inserted at the top of the box, which is important when, at the beginning of their project, students have to initialize the station. The power cable can be connected to one of the vertical sides of the box. The rectangular hole in Fig. 7.1.

- The screen and buttons were deliberately placed above the box to prevent it from moving as much as possible during use. In fact, with the old box, the buttons were on a vertical surface, which meant that each time a button was pressed, the box rotated or moved. This invariably affected the growth measurement via imaging.

7.1.2 Afterword

Finally, so that the project can still evolve in the future, we leave the modeling files, made using FreeCad[1].

[1] https://www.freecad.org/

FreeCad file:
https://uclouvain-my.sharepoint.com/:f:/g/personal/louis_colinet_student_uclouvain_be/EtktZSTNDBVBrsSa9NaLbnMBvc-s0ZKkodyd2T7TMqsheQ?e=9uv1jI

The project can thus be adapted in the event that new sensors are added, pipes need to be added to the base to increase the height, or other as-yet unknown modifications are made.

Regarding the price, we can calculate it based on the total weight of the prints. This is approximately 285 grams. Since the price of PLA plastic is approximately 16€/kg[2], this gives us a final price of approximately 4.56€ depending on the plastic purchased. This is significantly cheaper than the box used so far which comes to a price of around 15-20€[3].

[2]https://www.amazon.fr/dp/B0CDR8VZW5
[3]https://www.amazon.fr/dp/B07QXB166S

Chapter 8

Future improvements

Although the development of the station can be considered broadly complete, we are aware that it still has room for improvement. This chapter therefore presents several possible areas for improvement, without claiming to be exhaustive.

8.1 Image capture

Several areas for improvement are possible regarding the image capture and processing process by the PhenoHive workstation.

8.1.1 Image retrieval interface

Currently, image retrieval from the workstation is done via the `scp` command in a terminal. Although this method is functional, it is not intuitive for users unfamiliar with the command prompt. A possible improvement would be to develop a dedicated graphical interface or a web interface allowing users to view, download, or delete captured images in a user-friendly manner. Such an interface would enhance the system's accessibility, particularly in an educational context.

8.1.2 Image calibration optimization

The calibration of image processing parameters, used in particular for plant skeleton detection, could be optimized. Indeed, the current algorithm repeatedly recalculates identical steps, particularly skeleton generation, which leads to unnecessary redundancies. Although computation times remain acceptable on the Raspberry Pi, a partial rewrite of the code would make this part more efficient, further reducing calibration time.

8.1.3 Improvement through machine learning

In the medium to long term, it would be possible to use the images collected by the station to create an annotated database, enriched over time by students. This database could then be used to train deep learning models aimed at improving the detection of plant characteristics. Such models could increase measurement accuracy or enable the detection of specific cases (anomalies, water stress, etc.). Although current results are satisfactory, this approach would represent a natural evolution of the system towards more intelligent automated phenotyping.

8.2 Sensor improvement

Adding sensors is a key lever for enhancing the PhenoHive station, both in terms of measurement accuracy and the diversity of observable parameters. Two areas for improvement were identified during the project: the use of a capacitive sensor for soil moisture measurement, and the integration of a CO_2 sensor.

8.2.1 Replacing the resistive sensor with a capacitive sensor

The sensor currently used to measure soil moisture is based on a resistive principle. Although simple to integrate and inexpensive, this type of sensor has a significant limitation: the electrodes in direct contact with the soil tend to oxidize or erode over time, which gradually impairs the reliability of the measurements.

A more sustainable alternative is the use of a capacitive sensor. Unlike the resistive sensor, this one does not directly measure soil conductivity, but rather its dielectric permittivity. It works on the principle of a capacitor: when soil moisture increases, the dielectric constant of the material between the capacitor plates changes, which alters the set capacitance. Since the electrodes are not in direct contact with the moist soil, wear is considerably reduced, making these sensors more stable and reliable over the long term.

The choice of the resistive sensor for this project was justified by its immediate availability through Ms. Pelsser and its very low cost. However, it would be worthwhile to test a capacitive model in a future iteration to assess whether it brings real added value to the system, particularly for long-term measurements.

8.2.2 Integration of a carbon dioxide (CO_2) sensor

Throughout the project's development, the integration of a CO_2 sensor was discussed as a major, or even, according to some supervisors, the "Holy Grail", solution for low-cost phenotyping. Such a sensor makes it possible to measure the plant's photosynthetic activity indirectly, by monitoring CO_2 consumption in an enclosed space. It would then be a complementary indicator to the data already collected (growth, evapotranspiration, etc.).

However, the practical implementation of this type of sensor presents several obstacles:

- High cost: Reliable CO_2 sensors are often expensive, which runs counter to the project's low-cost philosophy.
- Integration complexity: To obtain relevant measurements, it is generally necessary to create a sealed environment around the leaf to detect small variations in CO_2 concentration. This requirement makes the mechanical design much more complex, particularly in terms of insulation, airflow control, and synchronization with image capture.

Further research would be necessary to identify a sensor that is accurate, affordable, and compatible with the existing architecture. A possible hybrid solution, such as a removable measuring chamber or a low-cost sensor assisted by external calibration, could represent an avenue of research.

8.3 Improvements to data management

Data management is a central aspect of the PhenoHive station's operation, both for the reliability of measurements and for its long-term sustainability. Currently, all collected data—images, measurement logs, and log files—are stored locally on the station's SD card, with no automatic cleaning strategy.

8.3.1 Implementation of an automatic archiving or deletion system

As described in Dylan Goffinet's dissertation [2], the station's estimated storage capacity would allow for autonomy of more than 220 days without saturation. While this duration is relatively comfortable, it does not guarantee absolute security.

A relevant improvement would be the implementation of a system for automatic deletion or archiving of the oldest data. This system could be based on a temporary retention policy: for example, all data (images, measurements, logs) older than 3 or 4 months would be automatically deleted or moved to a temporary folder (a "recycle bin") before final deletion.

Chapter 9

Conclusion

This work is a continuation of the PhenoHive project, whose objective is to provide a compact, intuitive, and low-cost phenotyping station to enhance the practical learning of plant physiology as part of the bioengineering curriculum. Given the material, logistical, and financial constraints limiting access to high-performance tools for plant monitoring, this station aims to provide each group of students with a personal, easy-to-use device that is robust enough to be used over an entire experimental period.

During this thesis, significant improvements were made to both the hardware and software. On the hardware side, the integration of new sensors (humidity, light) allows for a broader range of data collection, while the redesign of the casing makes the station more compact and robust. On the software side, updating the image processing environment with PlantCV, optimizing growth calculation parameters, adding a shadow detection module, and implementing an online database via InfluxDB and Grafana have enhanced the system's accuracy, flexibility, and user-friendliness. The user interface has also been redesigned to facilitate calibration, real-time visualization, and measurement tracking.

The results demonstrate that it is possible to design a reliable and scalable phenotyping tool for a total cost of around 65€, while maintaining measurement quality adapted to educational needs. This system promotes student autonomy and provides them with direct and continuous access to data in a usable and visually clear format.

This work also opens the door to other opportunities: improving image processing algorithms through machine learning, replacing or adding sensors to expand the parameters measured (CO_2, capacitive humidity), automating data archiving, and developing an installation and user manual to facilitate the project's dissemination.

Ultimately, this improved version of the PhenoHive station represents a further step toward the creation of democratic phenotyping tools in an educational context. It combines ease of use, rich measurement capabilities, and controlled costs, while providing a solid foundation for future innovations in teaching and research in plant physiology.

References

[1] L. Lemaire and M. Lallemand, "Innovation pédagogique dans le cadre des travaux pratiques de physiologie végétale : développement d'une station de phénotypage pour plante individuelle," Master's thesis, Faculté des bioingénieurs, Université catholique de Louvain, 2023, prom. : Draye, Xavier. [Online]. Available: http://hdl.handle.net/2078.1/thesis:40991

[2] D. Goffinet, "La plante connectée : An intuitive low-cost phenotyping station," Master's thesis, École polytechnique de Louvain, Université catholique de Louvain, 2024, prom. : Pelsser, Cristel. [Online]. Available: http://hdl.handle.net/2078.1/thesis:48702

[3] S. Lutts and X. Draye, "Lbir1251 – manuel des travaux pratiques 2024-2025," 2024, assistants : Nicolas Biot et Lola Leveau ; Chercheurs impliqués : Marco D'Agostino, Théo Degand et Corentin Defalque ; Visited on October 3, 2025. [Online]. Available: https://moodle.uclouvain.be/pluginfile.php/176995/mod_resource/content/20/LBIR1251_ManuelTP_2024-2025.pdf

[4] International Plant Phenotyping Network, "International plant phenotyping network – infrastructure map," visited November 10, 2025. [Online]. Available: https://www.plant-phenotyping.org/infrastructure_map

[5] HT Sensor, "Tal226 miniature parallel beam load cell," visited November 15, 2025. [Online]. Available: http://htc-sensor.com/products/142.html

[6] Wikipedia contributors, "Wheatstone bridge," visited November 15, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Wheatstone_bridge

[7] Maroc Produits, "Capteur humidité sol maroc," visited May 22, 2025. [Online]. Available: https://marocproduits.com/produit/capteur-humidite-sol-maroc/

[8] Learning About Electronics, "Calculatrice diviseur de tension," visited May 22, 2025. [Online]. Available: https://www.learningaboutelectronics.com/LesArticles/Calculatrice-diviseur-de-tension.php

[9] JCHL - Shenzhen Jing Chuang He Li Tech, "Gl5537a 25-35k datasheet," visited May 27, 2025. [Online]. Available: https://lcsc.com/datasheet/lcsc_datasheet_2410121312_JCHL-Shenzhen-Jing-Chuang-He-Li-Tech-GL5537A-25-35K-_C11309.pdf

[10] Université catholique de Louvain, "Bachelier en sciences de l'ingénieur, orientation bioingénieur - programme détaillé par bloc annuel," visited on October 3, 2025. [Online]. Available: https://uclouvain.be/prog-2025-bir1ba-programme_annual_blocks

[11] ——, "Biologie et physiologie végétale," visited on October 3, 2025. [Online]. Available: https://sites.uclouvain.be/archives-portail/cdc2023/cours-2023-lbir1251

[12] Wikipedia contributors, "Evapotranspiration," visited November 10, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Evapotranspiration

[13] Futura-Sciences, "Phénotype : qu'est-ce que c'est ?" visited November 10, 2025. [Online]. Available: https://www.futura-sciences.com/sante/definitions/biologie-phenotype-224/

[14] Université catholique de Louvain, "Plant phenotyping facility - rootphair (eli)," visited November 10, 2025. [Online]. Available: https://www.uclouvain.be/en/research-institutes/eli/elia/rootphair

[15] Wikipedia contributors, "Aeroponics," visited November 10, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Aeroponics

[16] Plant Systems Biology, VIB-UGent, "Wiwam – maize," visited November 10, 2025. [Online]. Available: https://www.psb.ugent.be/phenotyping/wiwam-maize

[17] ——, "Wiwam – arabidopsis," visited November 10, 2025. [Online]. Available: https://www.psb.ugent.be/phenotyping/wiwam-arabidopsis

[18] ——, "Phenovision," visited November 10, 2025. [Online]. Available: https://www.psb.ugent.be/phenotyping/phenovision

[19] University of Copenhagen (PLEN), "Plant phenolab – high-throughput phenotyping greenhouses in taastrup," visited November 10, 2025. [Online]. Available: https://plen.ku.dk/english/about/pfv/the-greenhouses-in-taastrup/phenolab/

[20] Electros, "Qu'est-ce qu'une jauge de contrainte, les types de jauges de contrainte, le schéma de câblage et leur application." visited November 15, 2025. [Online]. Available: https://electros.tomathouse.com/fr/kipia/chto-takoe-tenzodatchik

[21] SparkFun Electronics, "24-bit analog-to-digital converter (adc) for weigh scales," visited November 15, 2025. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[22] M. Cordier, P. Rasti, C. Torres, and D. Rousseau, "Affordable phenotyping at the edge for high-throughput detection of hypersensitive reaction involving cotyledon loss," *Plant Phenomics*, vol. 6, 2024, DOI: https://doi.org/10.34133/plantphenomics.0204.

[23] S. Wu and al., “Mvs-pheno: A portable and low-cost phenotyping platform for maize shoots using multiview stereo 3D reconstruction,” *Plant Phenomics*, vol. 2020, Jan. 2020, DOI: https://doi.org/10.34133/2020/1848437.

[24] Wikipedia contributors, “Influxdb,” visited November 17, 2025. [Online]. Available: https://en.wikipedia.org/wiki/InfluxDB

[25] InfluxData, “Influxdb — real-world ai starts with time series.” visited November 17, 2025. [Online]. Available: https://www.influxdata.com/

[26] Wikipédia contributors, “Pip (gestionnaire de paquets),” visited November 15, 2025. [Online]. Available: https://fr.wikipedia.org/wiki/Pip_(gestionnaire_de_paquets)

[27] Python Software Foundation, “configparser — configuration file parser,” visited November 15, 2025. [Online]. Available: https://docs.python.org/3/library/configparser.html

[28] PlantCV contributors, “Canny edge detection - plantcv,” visited November 30, 2025. [Online]. Available: https://plantcv.readthedocs.io/en/stable/canny_edge_detect/

[29] Wikipedia contributors, “Dice–sørensen coefficient,” visited April 3, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Dice-S%C3%B8rensen_coefficient

[30] A. Sanin, C. Sanderson, and B. C. Lovell, “Shadow detection: A survey and comparative evaluation of recent methods,” *Pattern Recognition*, vol. 45, no. 4, pp. 1684–1695, Apr. 2012, DOI: https://doi.org/10.1016/j.patcog.2011.10.001; Visited July 21, 2025.

[31] Centre d’information sur l’eau, “Qu’est-ce que le stress hydrique ? comment y répondre ?” august 12, 2025. [Online]. Available: https://www.cieau.com/eau-transition-ecologique/enjeux/quest-ce-que-le-stress-hydrique-comment-y-repondre/

[32] Microchip Technology Inc., “Mcp3004;mcp3008 2.7v 10-bit a/d converters with spi interface,” visited May 27, 2025. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MSLD/ProductDocuments/DataSheets/MCP3004-MCP3008-Data-Sheet-DS20001295.pdf

[33] PyPI, “Adafruit-mcp3008: Python code to use the mcp3008 analog to digital converter with a raspberry pi or beaglebone black,” visited May 22, 2025. [Online]. Available: https://pypi.org/project/Adafruit-MCP3008/1.0.2/

[34] FARO, “Comment mesure-t-on la lumière ? unités de mesure et luxmètre,” visited May 27, 2025. [Online]. Available: https://faro.es/fr/blog/comment-mesure-t-on-la-lumiere/

[35] Wikipédia contributors, “Lumen (unité),” visited May 27, 2025. [Online]. Available: https://fr.wikipedia.org/wiki/Lumen_(unit%C3%A9)

[36] ——, “Lux (unité),” visited May 27, 2025. [Online]. Available: https://fr.wikipedia.org/wiki/Lux_(unit%C3%A9)

[37] Magic_squirrel_hat, “Sensor ldr datasheet,” visited May 27, 2025. [Online]. Available: https://www.reddit.com/r/arduino/comments/170gd2l/sensor_ldr_datasheet/

[38] Ed Nieuw, “Photoresistors gl55-series comparison,” visited May 27, 2025. [Online]. Available: https://ednieuw.home.xs4all.nl/Woordklok/LDR/GL55xxLDRs.html

[39] Le Disrupteur Dimensionnel, “Ldr ou photoresistance,” visited May 27, 2025. [Online]. Available: https://ledisrupteurdimensionnel.com/arduino/ldr-ou-photoresistance/

[40] Calculatatrice Ultra, “Calculateur de flux lumineux et d’éclairement en fonction de la distance; formule en ligne calculator ultra,” visited May 27, 2025. [Online]. Available: https://www.calculatorultra.com/fr/tool/luminous-flux-illuminance-distance-calculator.html#gsc.tab=0

[41] LampesDirect.fr, “Comment choisir le bon angle de faisceau ?” visited May 27, 2025. [Online]. Available: https://www.lampesdirect.fr/blog/comment-choisir-bon-angle-faisceau?switch=5892864744046720988

[42] ngrok, Inc., “ngrok – secure tunnels to localhost,” visited March 8, 2025. [Online]. Available: https://ngrok.com/

[43] Malekal, “Systemd: Configuration et fonctionnement des services linux (daemon),” visited November 17, 2025. [Online]. Available: https://www.malekal.com/systemd-service-linux-configuration-et-fonctionnement-daemon/

[44] NSSM – the Non-Sucking Service Manager, “Nssm (non-sucking service manager),” visited March 8, 2025. [Online]. Available: https://nssm.cc/

[45] Wikipédia contributors, “.bat,” visited March 8, 2025. [Online]. Available: https://fr.wikipedia.org/wiki/.bat

APPENDICES

Appendix A

Electrical connection schematic

During the thesis, some changes were made to the circuit. This is why we took the diagram available in the appendices of Dylan's thesis [2] and modified it.

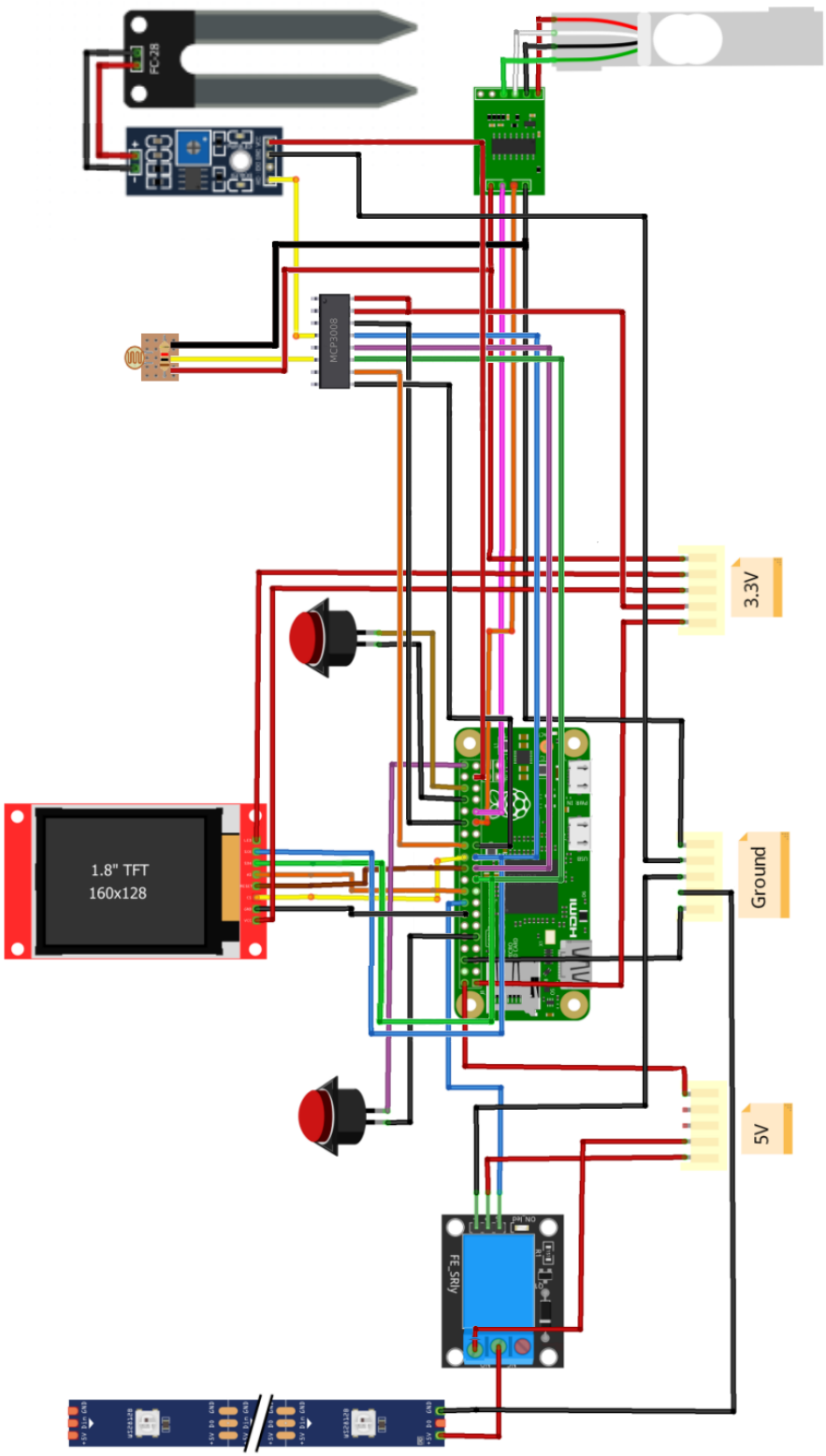
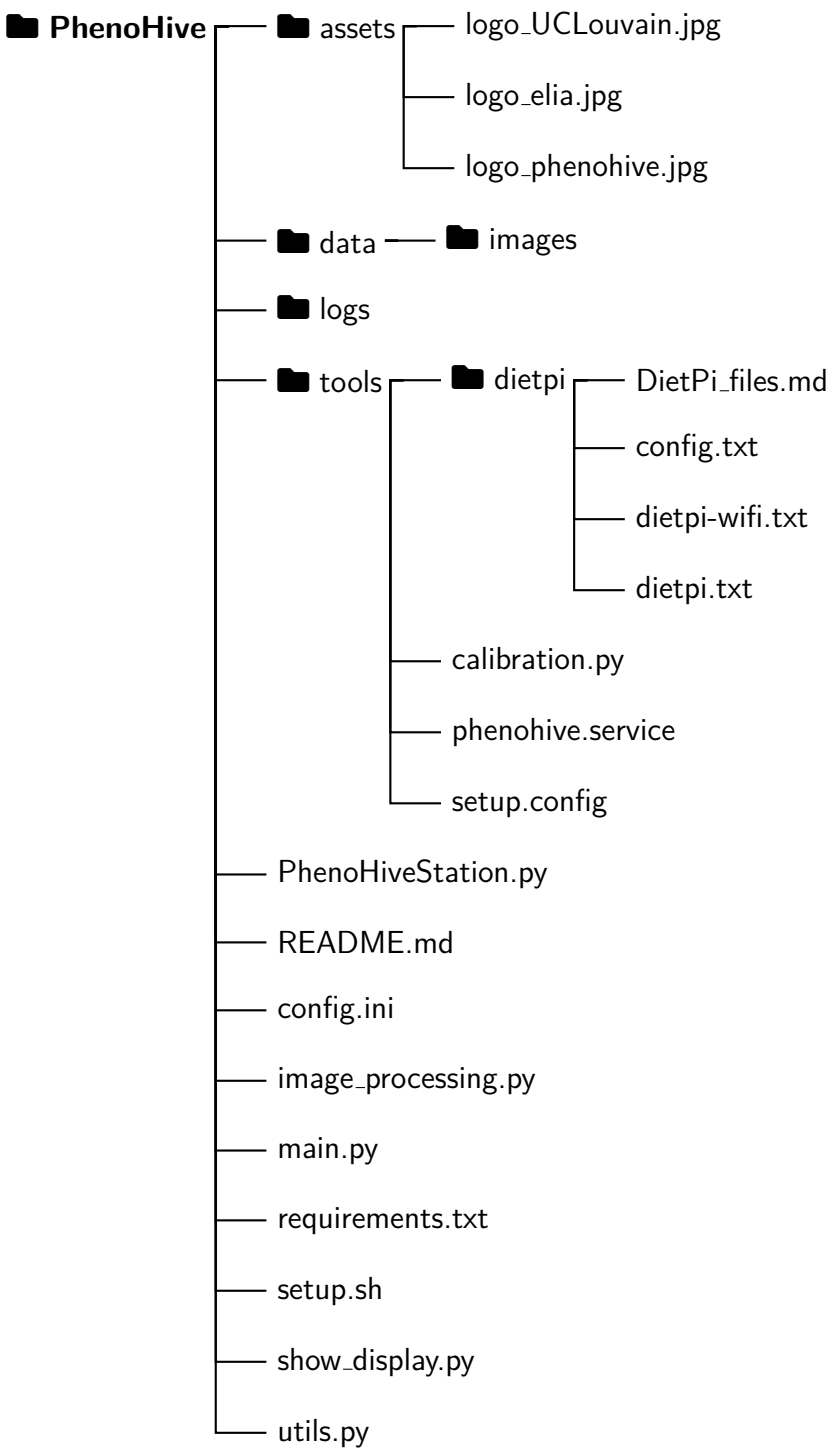


Figure A.1: Circuit connection diagram, modification of the pre-existing diagram in Dylan Goffinet's thesis [2]

Appendix B

Explanation of each project file in detail



Starting from top to bottom, we have:

- **assets**: This is simply a folder containing the various logos related to the project.

- **data**: This folder itself contains another folder **image** which will contain all the images taken during the measurements. Once the measurements are launched, the folder will also contain a **measurments.csv** file containing the weight and growth values.

- **logs**: This folder will contain daily text files containing information about the station's status and proper operation. An example is shown in Fig. B.1. Logs can have three forms: INFO, DEBUG, and ERROR. These files can be displayed using the command:

$$cat\ PhenoHive/logs/2025-04-04\_PhenoHive.log$$

  where the date must be changed to the desired date (note: date format: year-month-day). Finally, this logging system greatly aids debugging by allowing monitoring even when the station is running unattended.

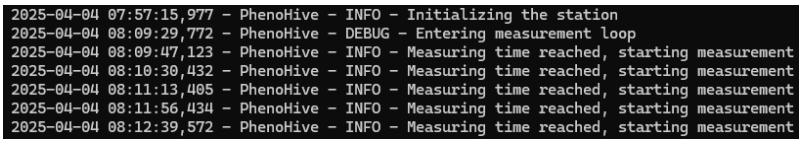


Figure B.1: Example of log file

- **tools/dietpi**: This folder contains three files useful for manually initializing the station. Unlike the initialization explained above, which is a copy, it is possible to initialize the Raspberry Pi directly. To do this, we have:

  - **config.txt**: In this file, we can enable the camera and the SPI (Serial Peripheral Interface).
  - **dietpi-wifi.txt**: This file allows us to pre-enter the Wi-Fi connection information. Here, the lines *aWIFI_SSID[0]='PhenoHive'* and *aWIFI_KEY[0]='PhenoHive'* are filled in so that our station immediately connects to our hotspot. This also enables SSH connection.
  - **dietpi.txt**: In this file, we can choose the raspberry timezone: *AUTO_SETUP_TIMEZONE=Europe/Brussels*, the network options that allow us to choose wifi rather than ethernet with: *AUTO_SETUP_NET_ETHERNET_ENABLED=0* and *AUTO_SETUP_NET_WIFI_ENABLED=1*, the hostname: *AUTO_SETUP_NET_HOSTNAME=PhenoHive*, the choice of SSH server, here Dropbear: *AUTO_SETUP_SSH_SERVER_INDEX=-2*, the system password: *AUTO_SETUP_GLOBAL_PASSWORD=phenohive*, software to install automatically and many other parameters.

  All these settings can be changed later using the Raspberry Pi configuration panel. This is accessible via the command:

$$dietpi - config$$

The last file we haven't discussed, **DietPi_files.md**, is a markdown file. This type of file allows for aesthetically pleasing rendering of text, often explanatory, about a folder, file, code, etc., in a project. In our case, it explains the changes that were made to the three files presented above.

- **calibration.py**: This file contains Python code to pre-calibrate the scale. However, it is also possible to calibrate it directly from the station. The latter method is simpler and more intuitive.
- **phenohive.service**: This file allows the initialization of the service specific to the station code. A service, also called a daemon, allows programs to run from startup and in the background [43]. This service therefore allows the main code of the station to be launched as soon as it is launched.
- **setup.config**: This file contains all the required packages that will need to be installed.
- **PhenoHiveStation.py**: This file contains the *PhenoHiveStation* class. In programming, a class is like a template used to create objects. It contains both data called attributes and functions called methods that describe the behavior of these objects. Our *PhenoHiveStation* class will therefore have, for example, attributes like *station_id*, *image_path*, *status*, and many others. It will also have functions like *get_weight*(), *save_photo*(), *measurement_pipeline*(), etc.
- **README.md**: This file is a Markdown file as explained previously. It provides information about the project in general, how to use the station, and how to install the station.
- **config.ini**: This file contains several variables that are specific to a station and must be kept in memory even when the station is turned off. It is divided into different sections such as Station, InfluDB, Paths etc. In these sections we find the variables:
  - **id**, which allows you to know which station is communicating with the database.
  - **token**, which allows you to identify yourself to the database.
  - The paths to the different folders where different data should be saved.
  - The different connection pins, for example, for the camera and the buttons.
  - **time_interval**, which defines the time between two plant measurements.
  - **calibration_weight** and **load_cell_cal** which are the reference weight for calibrating the scale and the factor for converting raw weight to weight in grams. This will be explained in more detail in the next section.
  - Parameters useful for image analysis.
- **image_processing.py**: This file contains the code for analyzing the image taken by the station. Its functions return a total measurement of the plant's skeleton, that is, a sum of the length of each leaf.

- **main.py**: This file is the heart of the station. It contains all the interface menus and interacts with the user through buttons or even with time to call the functions used at the time.
- **requirements.txt**: This file contains the Python packages that need to be installed with pip. pip is a package manager used to install and manage packages written in Python [26].
- **setup.sh**: This file allows for the more manual installation of the station. It uses the **setup.config** and **requirement.txt** files to determine which packages to install on the Raspberry Pi. This file also allows us to enable the SPI, the camera, and the service. To use it, enter the command:

  sudo bash setup.sh

  More details on this manual installation are available in the **README.md** file or in Dylan Goffinet's thesis [2].
- **show_display.py**: This file contains the functions that allow you to create the different screens to be displayed when the station is in operation.
- **utils.py**: This file contains three useful functions which are: *setup_logger()*, *create_folders()* and *save_to_csv()*

Appendix C

Detail and source of each price

This appendix includes all the sources used for pricing.

It's important to note that all the prices listed in this table are mostly for single components. For some components, a bulk order could drastically reduce the total cost.

As some sources are no longer available, other products have been used for this pricing.

Component	Quantity	Price (€)
Raspberry Pi Zero W with headers	1	19.95
Micro USB Power Supply (5V, 3A)	1	3.34
ST7735 LCD screen (128*160)	1	2.06
Button	2	0.29
LED lighting strips	1	0.69
Relay module KY-019	1	0.57
Picamera with Raspberry Pi Zero adapter	1	3.59
Tal226 load cell	1	11.69
HX711 converter	1	1.24
16 GB SD card	1	3.97
WAGO splicing connector with lever	3	0.59
Scale device (3D printing)	1	5
Cable (4 wires) 1m	1	0.88
Female to female cable (20 pieces)	1	0.64
Male to female cable (20 pieces)	1	0.64
Male to male cable (20 pieces)	1	0.64
TOTAL		57.13

Table C.1: Price of each element of the station

Sources:

-Raspberry Pi Zero W: https://shop.mchobby.be/en/motherboards/1892-raspberry-pi-zero-w-with-header-v11-3232100018921.html
-Power supply: https://nl.aliexpress.com/item/1005004113767773.html?gatewayAdapt=glo2nld
-LCD screen: https://nl.aliexpress.com/item/32843115817.html?gatewayAdapt=glo2nld
-Button: https://nl.aliexpress.com/item/1005003278096193.html?spm=a2g0o.productlist.main.138.265aDCT2DCT2VF&algo&gatewayAdapt=fra2nld
-LED: https://nl.aliexpress.com/item/1005006719406273.html?spm=a2g0o.productlist.main.1.11b751b7H4SrBk&algo_pvid=62495e70-100c-4c57-886a-c5a652bf214d&algo_exp_id=62495e70-100c-4c57-886a-c5a652bf214d-0&pdp_ext_f&gatewayAdapt=fra2nld
-Relay: https://nl.aliexpress.com/item/1005006719406273.html?spm=a2g0o.productlist.main.1.11b751b7H4SrBk&algo_pvid=62495e70-100c-4c57-886a-c5a652bf214d&algo_exp_id=62495e70-100c-4c57-886a-c5a652bf214d-0&pdp_ext_f&gatewayAdapt=fra2nld
-Picamera: https://nl.aliexpress.com/item/32901067278.html?gatewayAdapt=glo2nld
-Load cell: https://fr.aliexpress.com/item/32663837061.html?spm=a2g0o.productlist.main.5.da9d30c2AK6ZxS&algo_pvid=0edb8bde-ea97-4540-bf4f-c900da11661b&algo_exp_id=0edb8bde-ea97-4540-bf4f-c900da11661b-4&pdp_ext_f
-HX711 converter: https://fr.aliexpress.com/item/1005006574602903.html?spm=a2g0o.productlist.main.6.3c14387cn3jWx0&algo_pvid=ef6e3af5-4163-4611-ac70-d3e815098066&algo_exp_id=ef6e3af5-4163-4611-ac70-d3e815098066-5&pdp_ext_f
-SD card: https://nl.aliexpress.com/item/1005003189934704.html?gatewayAdapt=glo2nld
-WAGO: https://fr.aliexpress.com/item/1005008637221489.html?spm=a2g0o.productlist.main.2.3f926de0dVKYsW&algo_pvid=dff270ec-bd99-433d-b358-fb4beb52a75b&algo_exp_id=dff270ec-bd99-433d-b358-fb4beb52a75b-1&pdp_ext_f
-Cable 4 wires: https://fr.aliexpress.com/item/1005001695393491.html?spm=a2g0o.productlist.main.3.4f4155c5Xn90Xn&algo_pvid=8c1c7e0f-c1c1-4be1-a35c-44cf7539f828&algo_exp_id=8c1c7e0f-c1c1-4be1-a35c-44cf7539f828-2&pdp_ext_f
-Others cable: https://nl.aliexpress.com/item/4000323249974.html?gatewayAdapt=glo2nld

Appendix D

Initialisation and user manual

In order to be able to edit the document in the future, you will find the LaTeX file and images by following this link:

Latex files:
https://uclouvain-my.sharepoint.com/:f:/g/personal/louis_colinet_student_uclouvain_be/EtktZSTNDBVBrsSa9NaLbnMBvc-s0ZKkodyd2T7TMqsheQ?e=9uv1jI

Manuel d'installation et d'utilisation
Station PhenoHive

Louis Colinet

Août 2025

Ce document fournit les instructions essentielles pour l'initialisation et l'utilisation de la station **PhenoHive**. Ce document est séparé en trois parties:

- La présentation de la station
- L'initialisation de la station.
- L'utilisation de l'interface graphique et des menus.

1 Présentation de la station

La station **PhenoHive** est un dispositif compact et autonome destiné à l'observation et à la mesure de paramètres physiologiques d'une plante. Elle intègre plusieurs capteurs et éléments matériels permettant d'effectuer un suivi complet. Les principaux composants sont présentés ci-dessous. La figure 1, 2 et 3 permettent de visualiser ces éléments.

- **Écran et boutons**: L'écran intégré permet d'afficher les menus et les mesures en temps réel. Les boutons physiques situés à côté de l'écran servent à naviguer dans l'interface et à lancer certaines actions comme la calibration ou le démarrage d'une acquisition.
- **Caméra**: Une caméra miniature (*Picamera*) prend des images de la plante à intervalles réguliers. Ces images sont ensuite traitées via `PlantCV` pour estimer la croissance. La caméra est fixée dans la station afin d'être stable et de conserver le même cadrage.
- **Balance**: La balance est constituée d'une cellule de charge reliée à un convertisseur analogique-numérique. Elle permet de mesurer avec précision la masse du pot et de la plante, ce qui autorise le suivi de l'évapotranspiration.
- **Capteur de lumière**: Un capteur de lumière mesure l'intensité lumineuse incidente sur la plante. Les données peuvent être utilisées pour corréler la croissance avec l'exposition lumineuse. Le capteur est placé à proximité de la plante, en position fixe.

- **Capteur d'humidité**: Le capteur d'humidité du sol permet d'estimer la teneur en eau du substrat. Il est inséré directement dans le terreau pour obtenir une mesure continue. Cette information complète les données de poids pour évaluer la transpiration et les besoins en arrosage.
- **Pied télescopique**: Le pied télescopique supporte la station qui intègre la caméra et permet d'ajuster sa hauteur en fonction de la taille de la plante. Cela garantit une qualité d'image tout au long de l'expérience.
- **Accès carte SD et alimentation**: Un accès direct au logement de la carte microSD permet d'initialiser le système. Un connecteur d'alimentation est accessible sur le boîtier pour brancher facilement l'adaptateur secteur.

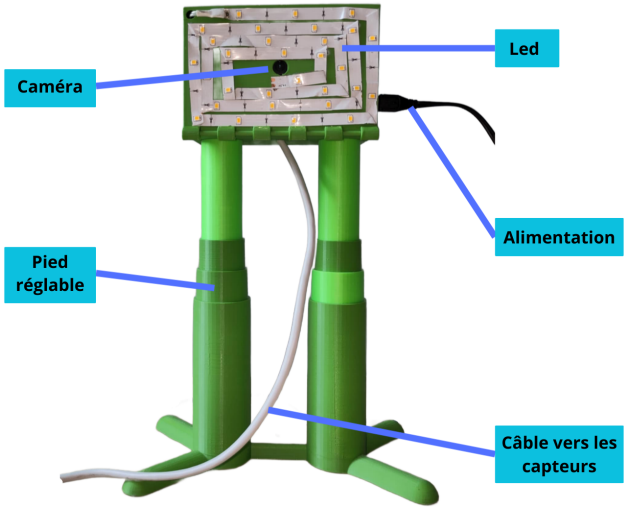


Figure 1: Station vue de face

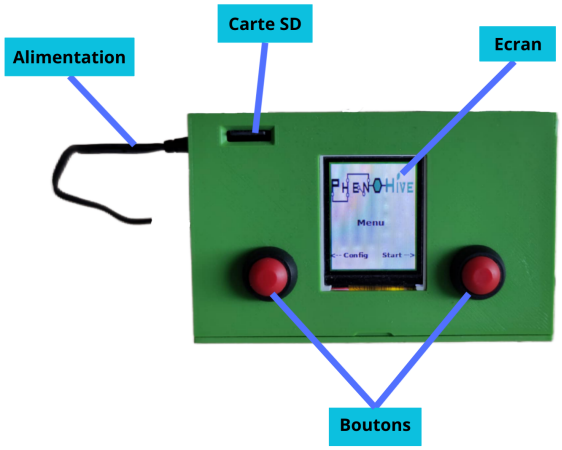


Figure 2: Station vue du dessus

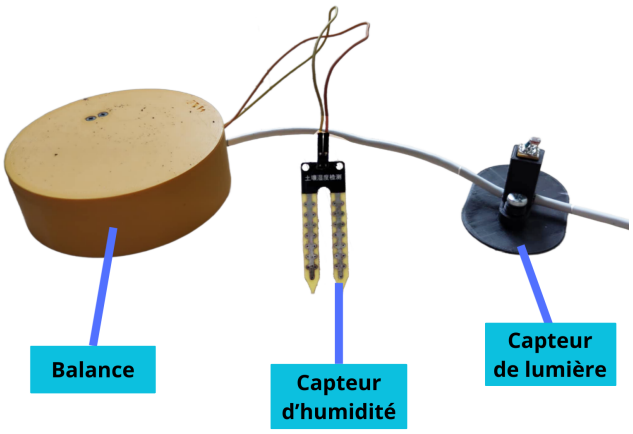


Figure 3: Capteurs en dehors de la station

2 Initialisation de la station

Pour initialiser la station PhenoHive, il est nécessaire de préparer la carte microSD contenant le système et la configuration du projet. Les étapes sont les suivantes :

Étape 1 : La première étape consiste à télécharger l'image de la carte SD[1] contenant le projet et les fichiers de configuration de la Raspberry Pi. Elle est disponible au lien suivant :

`https://uclouvain-my.sharepoint.com/:f:/g/personal/louis_colinet_student_uclouvain_be/EtktZSTNDBVBrsSa9NaLbnMBvc-s0ZKkodyd2T7TMqsheQ?e=9uv1jI`

Étape 2 : À l'aide de Balena Etcher qui est un logiciel libre permettant d'écrire des fichiers image disque, tels que des ISOs ou IMGs, sur des supports amovibles comme des clés USB ou des cartes SD[2], copier l'image téléchargée à l'étape 1 sur la carte SD.

Étape 3 : Avant d'insérer la carte SD dans la Raspberry Pi, il faut configurer la connexion Wi-Fi. Pour cela, ouvrir le fichier `dietpi-wifi.txt` présent à la racine de la carte SD et renseigner :

aWIFI_SSID[0]='NomDuReseau'
aWIFI_KEY[0]='MotDePasse'

Cette configuration permettra à la station de se connecter automatiquement sur réseau Wi-Fi.

Étape 4 : Insérer la carte SD dans le logement prévu sur la Raspberry Pi. Brancher ensuite l'alimentation. La station devrait démarrer automatiquement et se connecter au réseau Wi-Fi configuré.

[1] Une image mémoire est une copie bit à bit d'un support mémoire existant vers un autre support récepteur.
[2] Téléchargement : `https://etcher.balena.io/#download-etcher`

Étape 5 : Mettre à jour la station. Pour cela il faut se connecter en SSH à la station. Des explications plus précises sont données dans l'annexe -Ajouter l'annexe de Dylan Goffinet-. Une fois connecté, naviguer jusqu'au dossier contenant les fichiers du projet avec la commande:

$$cd\ PhenoHive$$

Puis, comme le projet est hébergé sur *GitHub*[3], mettre à jour les fichiers avec :

$$git\ pull$$

Étape 6 : Après la mise à jour, redémarrer la station pour prendre en compte les changements avec :

$$reboot$$

3 Utilisation de la station

Dans cette section, nous allons vous montrer comment utiliser l'interface directe de la station. Nous verrons comment calibrer la balance, comment positionner correctement la caméra par rapport à la plante, comment démarrer la prise de mesures et comment consulter l'état de la station. Pour faciliter la compréhension des différents chemins de navigation entre les menus, un schéma est présenté en Fig. 4.

Menu
Au démarrage de la station, on arrive sur le menu principal, intitulé "Menu". Chaque menu propose au maximum deux choix, sélectionnables grâce aux deux boutons situés à droite et à gauche de l'écran. Dans ce menu, on peut soit accéder au menu "Configuration", soit lancer les mesures.

Configuration
Depuis le menu principal, on accède au menu "Configuration". Celui-ci propose deux options: le bouton de droite conduit au menu de calibration de la balance ou des paramètres image, et le bouton de gauche au menu "Preview".

Calibration
Dans ce menu, deux choix sont possibles, naviguer vers la calibration de la balance ou vers la calibration des paramètres image.

Calibration de la balance
Ce menu sert à calibrer un coefficient de proportionnalité pour la balance. En effet, lorsque la station mesure le poids sur la balance, elle reçoit une valeur brute qui n'est pas en grammes. Pour afficher ce poids en grammes, il faut calculer un coefficient selon la formule :

$$poids\ en\ grammes = coeff \times poids\ brut$$

[3] *GitHub* est une plateforme en ligne permettant de stocker, partager et collaborer sur du code en utilisant *Git*, un outil de gestion de versions.

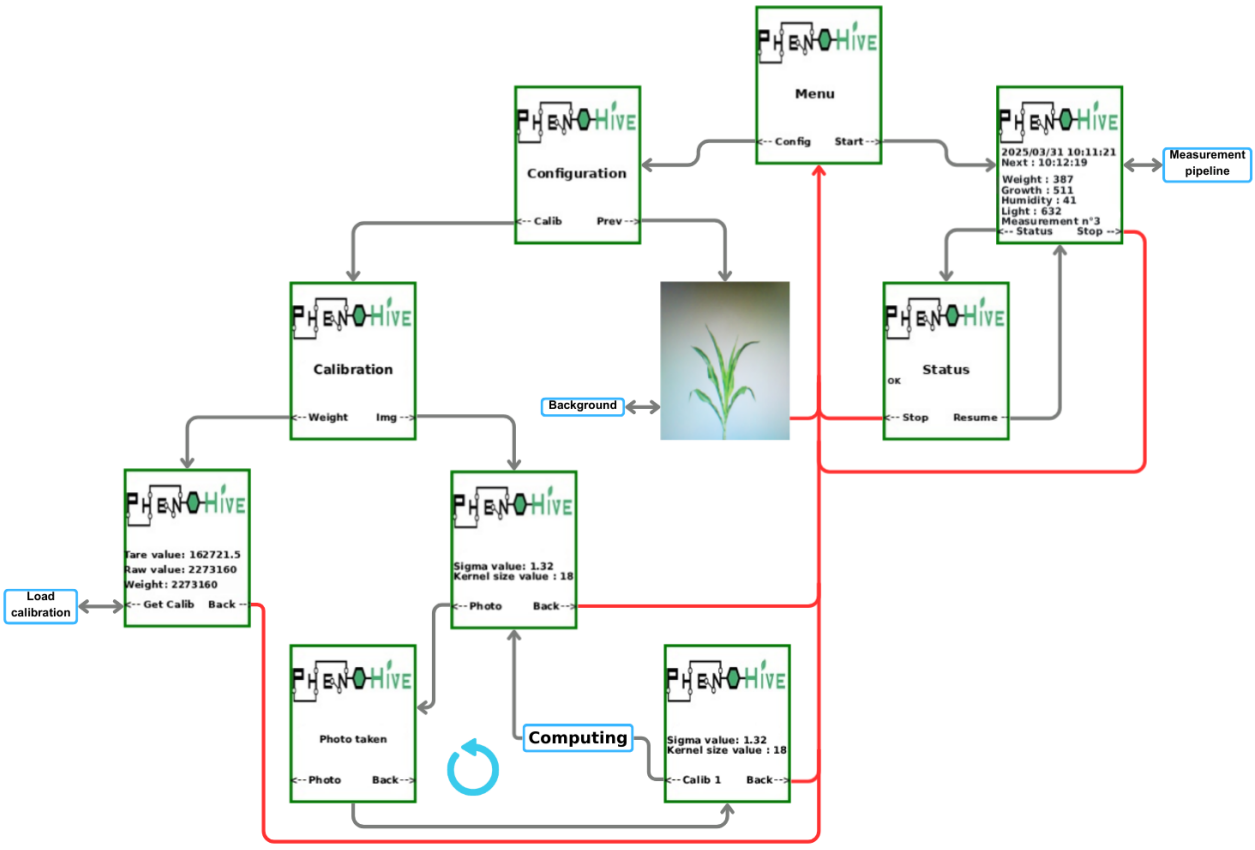


Figure 4: Schéma des navigations possibles pour chaque menu

La procédure est la suivante : avant d'entrer dans le menu de calibration de la balance (en appuyant sur "Weight" dans le menu "Calibration"), il faut s'assurer qu'aucun objet n'est posé sur la balance. Cela permet au programme d'enregistrer la tare dès l'entrée dans le menu. Une fois dans le menu, placer un poids de référence sur la balance (par défaut : 1,5 kg), puis appuyer sur "Get calib". Le programme calcule alors le coefficient avec :

$$coeff = \frac{poids\ ref}{poids\ brut - tare}$$

Pour vérifier la calibration, on peut appuyer plusieurs fois sur "Get calib" afin de contrôler que la valeur brute ne fluctue pas trop. À partir de la quatrième pression, le bouton "Get calib" devient un bouton de test. Un autre poids connu peut alors être placé sur la balance pour vérifier qu'il est correctement pesé. La tare, le poids de référence et le coefficient sont enregistrés dans la station, évitant ainsi de devoir recalibrer la balance à chaque démarrage.

Une fois la calibration terminée, on peut utiliser le bouton gauche "Back" pour revenir directement au menu principal.

Aperçu

Depuis le menu "Configuration", on peut accéder au menu "Preview" qui affiche en temps réel l'image capturée par la caméra. Cela permet de vérifier que la plante est correctement positionnée dans le champ de vision. Le bouton de droite permet de revenir au menu princi-

pal. Le bouton de gauche permet d'enregistrer l'arrière-plan. Cela signifie que l'on capture une image de la zone située derrière la plante. Pour ce faire, après avoir installé le setup et positionné la plante de manière à ce qu'elle soit visible par la caméra, retirez-la puis appuyez sur le bouton gauche. Il est essentiel de réaliser cette étape avant toute prise de mesure ou calibration des paramètres d'image, sinon la station renverra une erreur.

Calibration des paramètres image

L'algorithme d'analyse d'images utilisé pour mesurer la croissance comporte plusieurs paramètres réglables qui influencent fortement la qualité des résultats.

Le traitement des images suit les étapes suivantes :

- Conversion de la photo couleur en image en niveaux de gris.
- Détection des contours de la plante via un filtre de Canny.
- Remplissage des contours et calcul du squelette.
- Mesure de la longueur de ce squelette (en pixels), représentant la croissance.

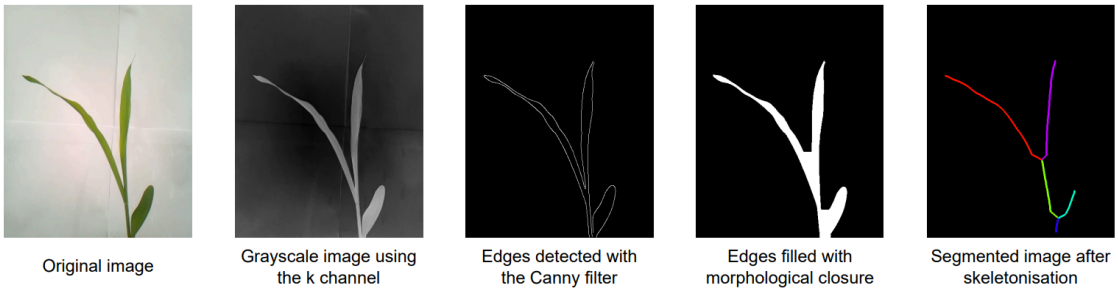


Figure 5: Pipeline d'analyse d'images utilisé pour la croissance (image provenant de D. Goffinet, "La plante connectée: An intuitive low-cost phenotyping station")

Deux paramètres sont particulièrement déterminants :

- sigma (filtre de Canny) : écart-type du filtre.

  Valeur faible → détection fine des détails mais risque de bruit.

  Valeur élevée → contours lissés et réduction des détails, voire absence de contours si trop élevée.

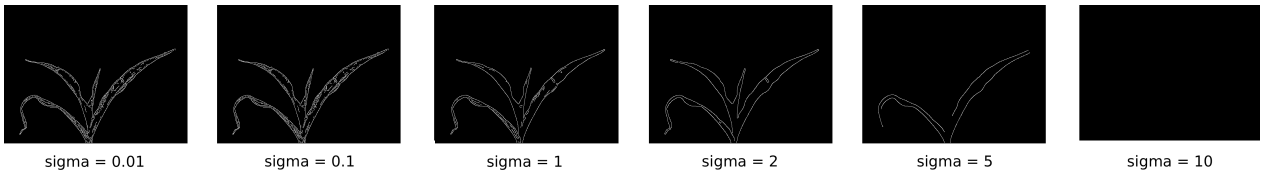


Figure 6: Effet de l'augmentation du paramètre sigma dans l'algorithme

- kernel_size (remplissage) : taille du noyau utilisé pour l'opération morphologique de fermeture.

  Valeur grande → comble de larges discontinuités mais risque de déformer les petits détails.

  Valeur petite → laisse subsister certaines interruptions dans les contours.

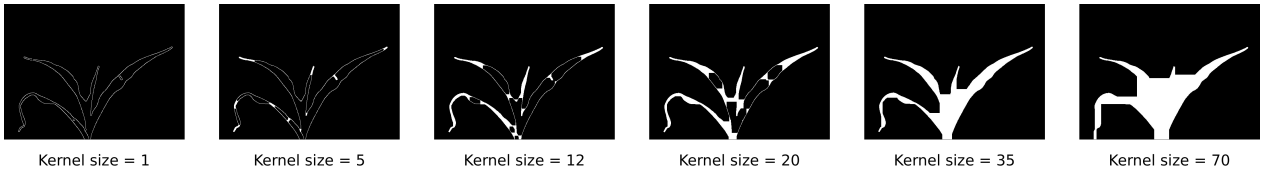


Figure 7: Effet de l'augmentation du paramètre kernel_size dans l'algorithme

Ces paramètres doivent être ajustés en fonction de la qualité et du contraste des images pour optimiser la détection du squelette de la plante.

Finalement, voici la procédure concrète de calibration des paramètres :

- Prendre une photo de calibration:

  Dans le menu, le bouton de gauche s'intitule "photo". Appuyer dessus prend une photo nommée *img_ calib.jpg*. Il faut ensuite récupérer cette photo sur votre ordinateur avec la commande :

  ```
  scp root@192.168.1.15:~/PhenoHive/data/images/img_calib.jpg
  C:\Users\path to folder
  ```

  Remplacez 192.168.1.15 par l'adresse IP de votre station (trouvable avec nmap[4]).

- Créer un squelette de référence:

  Ouvrir *img_ calib.jpg* dans Photopea et tracer manuellement le squelette de la plante. Ensuite, sauvegarder l'image sous le nom *skeleton_ ref.jpg*. L'annexe -Ajouter l'annexe sur Photopea- explique plus en détail cette étape. Finalement, envoyer ce fichier vers la station avec :

  ```
  scp C:\Users\path to folder\skeleton_ref.jpg
  root@192.168.1.15:~/PhenoHive/data/images/
  ```

- Lancer l'algorithme de calibration:

  Appuyer sur "test 1", ensuite, l'algorithme testera automatiquement plusieurs combinaisons de paramètres (sigma et kernel_size) et choisira ceux qui donnent le meilleur résultat. Après chaque test, les paramètres sont affinés, c'est pourquoi vous pouvez répéter la procédure de test si nécessaire pour obtenir une calibration optimale.

Mesures
Le menu des mesures affiche :

[4]https://nmap.org/

- la date et l'heure actuelles;
- l'heure de la prochaine mesure;
- le poids, la croissance, la lumière et l'humidité mesurés lors de la dernière acquisition;
- le nombre total de mesures depuis le dernier arrêt.

Lorsqu'une mesure est déclenchée, le programme suit un enchaînement d'étapes appelé *pipeline* de mesure (Fig. 8) :

1. Prendre une photo et l'afficher pour vérifier le cadrage de la plante.
2. Analyser l'image pour calculer la croissance totale des feuilles (en pixels).
3. Mesurer le poids et afficher la valeur (en grammes).
4. Mesurer la lumière et afficher la valeur (en Lux).
5. Mesurer l'humidité et afficher la valeur (en pourcentage de masse d'eau).
6. Envoyer toutes les données collectées vers la base InfluxDB via Wi-Fi.

Puis le programme revient au menu "Mesures" jusqu'à la prochaine acquisition.

Le bouton "Stop" permet d'interrompre les mesures et de revenir au menu principal. Un autre bouton permet d'accéder au menu "Statut".

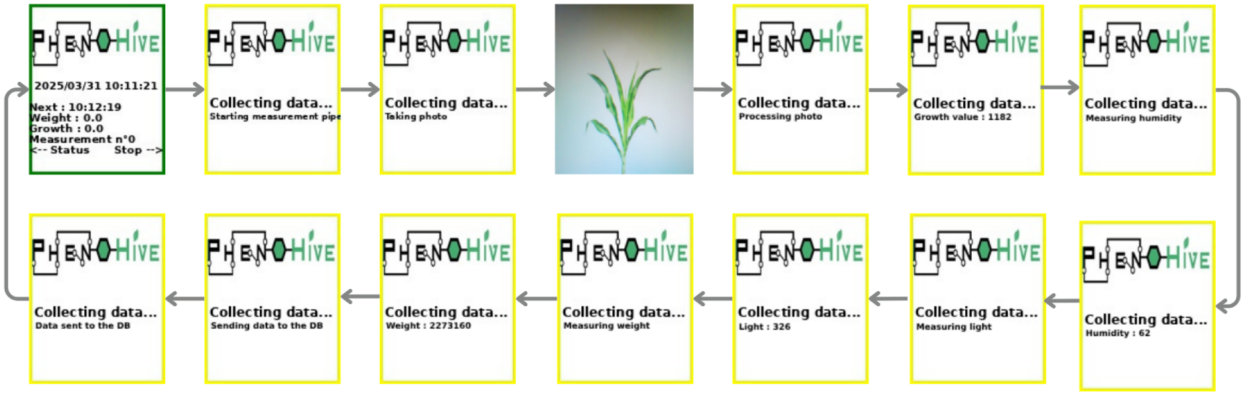


Figure 8: Pipeline de mesure

Statut

Le menu "Statut" affiche le type d'erreur survenue, utile lorsque la station fonctionne sans surveillance. En l'absence de problème, le statut affiche simplement "ok" (Fig. 4). Depuis ce menu, on peut arrêter complètement les mesures et revenir au menu principal, ou retourner au menu "Mesures" pour reprendre.

Les bordures colorées

L'écran peut afficher des bordures de couleur, correspondant à l'état actuel de la station :

Vert : fonctionnement normal, aucun problème détecté.

Rouge : une erreur est survenue (plante mal positionnée, connexion à la balance perdue, caméra défectueuse ou obstruée, etc.).

Bleu foncé : fonctionnement normal mais la base de données n'est pas connectée. La station enregistre alors localement ses données.

Jaune : la station est en cours d'exécution du pipeline de mesure.

Grâce à ces bordures, l'utilisateur peut connaître l'état de la station d'un simple coup d'œil.

4 Conseil pratique

- Lors de l'installation de la station, privilégiez si possible un emplacement où aucune ombre ne viendra interférer. Les mesures n'en seront que meilleures.
- Les pieds sont télescopiques et donc réglables. Pour monter la station, vous devez simplement dévisser puis revisser les vis présentes dessus. Attention, il n'est pas nécessaire de les visser au tournevis, les tourner à la main suffit. Un serrage trop fort pourrait endommager le plastique.
- Il est possible dans Grafana de poser des annotations sur les courbes. Pour cela, vous pouvez simplement cliquer dessus et écrire ce que vous voulez. Cela est pratique pour noter les arrosages par exemple.
- Tout au long de l'expérience, vous serez amenés à déplacer la station pour garder le plant de maïs dans le champ de la caméra. Cela affectera la courbe de croissance mais ce n'est pas grave. Il faut garder à l'esprit que la mesure est relative et en pixels donc pas en centimètres. De plus, pour qu'une grosse variation soit comptabilisée, il faut attendre deux minutes sinon elle sera ignorée. Pour éviter cela, nous vous invitons, lorsque vous devrez modifier le setup, à sortir du menu "Mesure" et à recalibrer le champ dans le menu "Preview". Il est également conseillé, à chaque changement de plan de recapturer un background et de recalculer les paramètres image. Comme pour les arrosages, vous pouvez noter ces changements dans Grafana.
- Lorsque la station redémarre après un arrêt de l'alimentation, par exemple une panne de courant, une valeur anormalement haute d'humidité sera mesurée. Ceci est normal et se stabilisera plus tard à la bonne valeur.
- Dans certains menus comme "Preview", il faut maintenir les boutons au moins 2 secondes pour avoir une réaction.

Appendix E

Software and motherboard comparison tables

E.1 Methodology for constructing comparative tables

To develop the comparison tables (software and electronic boards), we used various sources cited below. The criteria chosen (accuracy, ease of use, required hardware resources, flexibility, cost, etc.) were defined based on the specific requirements of the phenotyping project and information gathered from various sources. The information collected was then synthesized to obtain a clear and coherent overview, facilitating the selection of the most appropriate solution for the project.

Sources used for software

- **PlantCV**:
  `https://plantcv.org/`
  `https://pubmed.ncbi.nlm.nih.gov/29209576/`
- **ImageJ/Fiji**:
  `https://arxiv.org/abs/1701.05940`
  `https://imagej.net/software/fiji/`
- **OpenCV**:
  `https://opencv.org/`
  `https://www.machinelearningmastery.com/a-gentle-introduction-to-opencv-an-open-source-library-for-computer-vision-and-machine-learning/`
  `https://www.geeksforgeeks.org/opencv-python-tutorial/`

https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

- **ilastik**:
  https://en.wikipedia.org/wiki/Ilastik
  https://www.nature.com/articles/s41592-019-0582-9
  https://mpicbg-scicomp.github.io/ipf_howtoguides/guides/Ilastik_PixelClassification.html
- **Phenotiki**:
  https://phenotiki.com/
  https://www.quantitative-plant.org/software/phenotiki
  https://iris.sssup.it/retrieve/dd9e0b31-f6c6-709e-e053-3705fe0a83fd/Minervini_et_al-2017-The_Plant_Journal.pdf
  https://www.turing.ac.uk/news/publications/phenotiki-open-software-and-hardware-platform-affordable-and-easy-image-based
- **PhenoImage**:
  https://digitalcommons.unl.edu/csearticles/267/
  https://cse.unl.edu/~yu/homepage/software/PhenoImage/index.html
  https://acsess.onlinelibrary.wiley.com/doi/abs/10.1002/ppj2.20015
  https://www.biorxiv.org/content/10.1101/2020.09.01.278234v1
  https://pmc.ncbi.nlm.nih.gov/articles/PMC9590503/
- **DeepPlantPheno**:
  https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2017.01190/full
  https://github.com/p2irc/deepplantphenomics
- **piCore**:
  https://tinycorelinux.net/5.x/armv6/releases/README
  https://yoppychia.wordpress.com/2017/05/29/raspberry-pi-installing-picore-9-0/
  https://iotbytes.wordpress.com/picore-tiny-core-linux-on-raspberry-pi/

E.1.1 Sources used for motherboard

- **Raspberry Pi 4 Model B**:
  https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
  https://www.pishop.us/product-category/raspberry-pi/raspberry-pi-boards/current-pi-boards/
  https://en.wikipedia.org/wiki/Raspberry_Pi
  https://fr.wikipedia.org/wiki/Raspberry_Pi https://pubmed.ncbi.nlm.nih.gov/29209576/

- **Raspberry Pi Zero 2 W**:
  https://en.wikipedia.org/wiki/Raspberry_Pi
  https://www.raspberrypi.com/products/
  https://shop.mchobby.be/fr/cartes-meres/2334-raspberry-pi-zero-2-w-avec-header-wireless-cam-port-3232100023345.html
- **NVIDIA Jetson Nano**:
  https://dronebotworkshop.com/nvidia-jetson-developer-kit/
  https://www.tomshardware.com/news/jetson-nano-features-price%2C38856.html
- **BeagleBone Black**:
  https://www.beagleboard.org/boards/beaglebone-black
  https://docs.beagleboard.org/boards/beaglebone/black/ch04.html
  https://www.renaissancerobotics.com/BeagleboneBlack.html
- **Odroid C4**:
  https://www.hardkernel.com/shop/odroid-c4/
  https://www.ebay.com/itm/326461964166
  https://www.reddit.com/r/hardware/comments/g6q5dk/50_odroidc4_raspberry_pi_4_competitor_combines/
- **Arduino Portenta H7**:
  https://store-usa.arduino.cc/collections/portenta-family https://www.wired.com/beyond-the-beyond/2020/01/powerful-new-arduino-portenta-family
  https://docs.arduino.cc/hardware/portenta-h7
  https://store.arduino.cc/products/portenta-h7?srsltid=AfmBOood5XMe-J61fexQPDp5l27SOhLLayvM13jS5O3EeI_pWLbRFJOi
  https://www.wevolver.com/specs/arduino-portenta-h7

E.2 Software comparison

Software	**Analysis Accuracy**	**Ease of Use**	**Hardware Resources Required**	**Flexibility**	**Cost**
PlantCV	High accuracy, specially designed for plant phenotyping	Medium: requires Python knowledge	Low: Runs well on Raspberry Pi Zero	Very flexible, editable in Python	Free
ImageJ/Fiji	Very accurate, but depends on plugins and configured settings	Easy: accessible GUI	Medium: Requires a PC with standard capabilities	Very flexible with many plugins	Free
OpenCV	Highly accurate for computer vision, wide range of tools	Difficult: requires programming skills (Python, C++)	Medium: Works on Raspberry Pi but requires optimization	Extremely flexible	Free
ilastik	Good accuracy, based on machine learning	Easy: Intuitive interface, minimal learning required	Medium: Runs on standard PCs without a GPU	Moderately flexible	Free
Phenotiki	Very high accuracy thanks to deep learning	Average: Requires initial setup	High: Requires a PC with a GPU for best performance	Moderate flexibility, integration possible	Free
PhenoImage	Accuracy depends on the quality of pipeline configuration	Medium: Requires basic computer vision skills	Medium to High: Runs best on high-performance PCs	Flexible with the right integrations	Free
DeepPlantPheno	Very high accuracy thanks to deep learning	Medium to difficult: requires expertise to configure and train the models	High: Requires a PC with a GPU for best performance	Very flexible with suitable models	Free
piCore	Depends on the tools used with this system	Medium: Complex basic configuration	Very low: designed for resource-limited devices	Very flexible when coupled with other tools (OpenCV, etc)	Free

Table E.1: Comparison of software for plant phenotyping

E.3 Motherboard comparison

Circuit	**Estimated Price**	**Performance**	**Ease of Integration**	**Community Support**	**Built-in WiFi/Bluetooth**
Raspberry Pi 4 Model B	50-100€	Very High	Very Easy	Excellent	Yes
Raspberry Pi Zero 2 W	20-25€	Average	Very Easy	Excellent	Yes
NVIDIA Jetson Nano	100-120€	Very High	Average	Good	No (module required)
BeagleBone Black	50-60€	Average	Average	Good	No (module required)
Odroid C4	60-70€	High	Average	Average	No (module required)
Arduino Portenta H7	80-100€	High	Medium	Good	Yes

Table E.2: Comparison of circuits

Appendix F

Transformation of the calibration image into a reference skeleton

In this appendix, we explain the steps to transform the image taken by the station called *img_calib.jpg* into a reference skeleton called *skeleton_ref*. To do this, we will use Photopea[1]. Photopea is a free online image editor that works directly in your browser, without installation. It offers advanced features similar to Photoshop, such as layers, masks, filters, and retouching tools. We can note that the steps performed are also possible on other image editors.

Import the image into Photopea

Once the image is imported from the station to your computer, you need to import it into Photopea. To do this, go to *Fichier* > *Ouvrir* then select the image from the file in which it was saved. You can also simply drag the photo directly onto the site.

[1] https://www.photopea.com/

Figure F.1: Photopea Main Menu

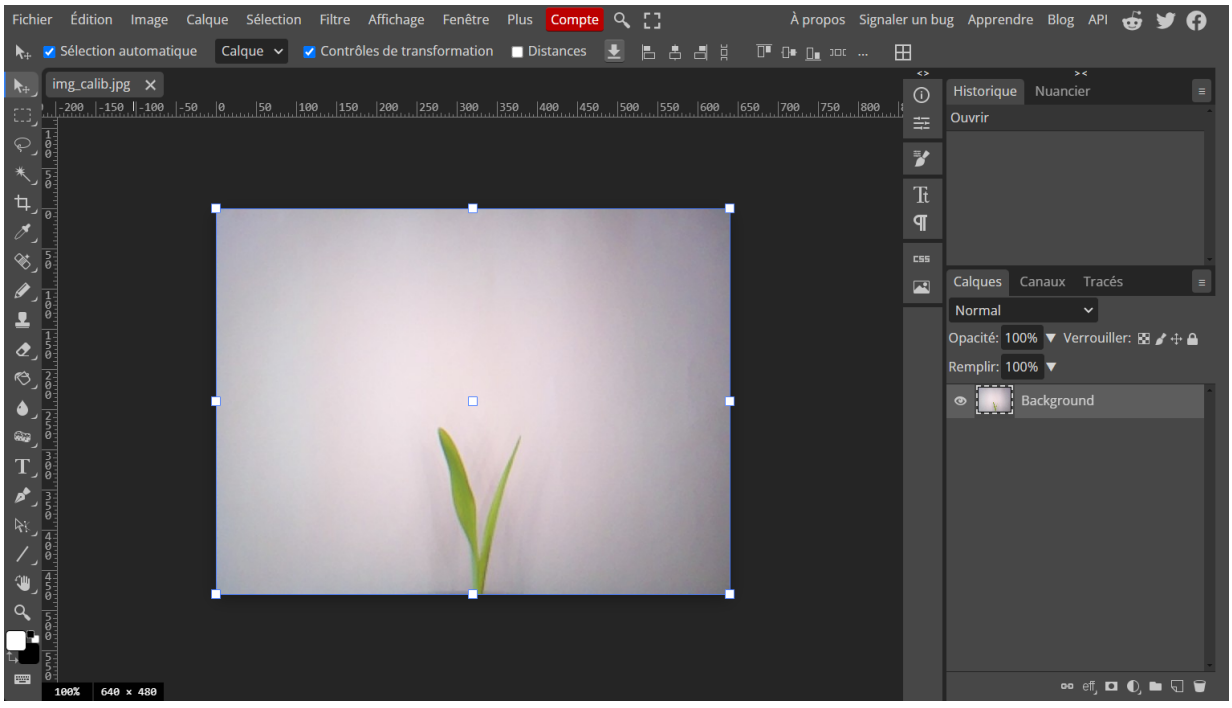


Figure F.2: Importation of img_calib

Adding a layer over the image

Once the import is complete, we can create a layer over the image. This will allow us to make changes without touching the original image. To do this, go to *Calque* > *Nouveau* > *Calque*. A new layer will then appear in the right-hand bar. Make sure that it is above the original image.

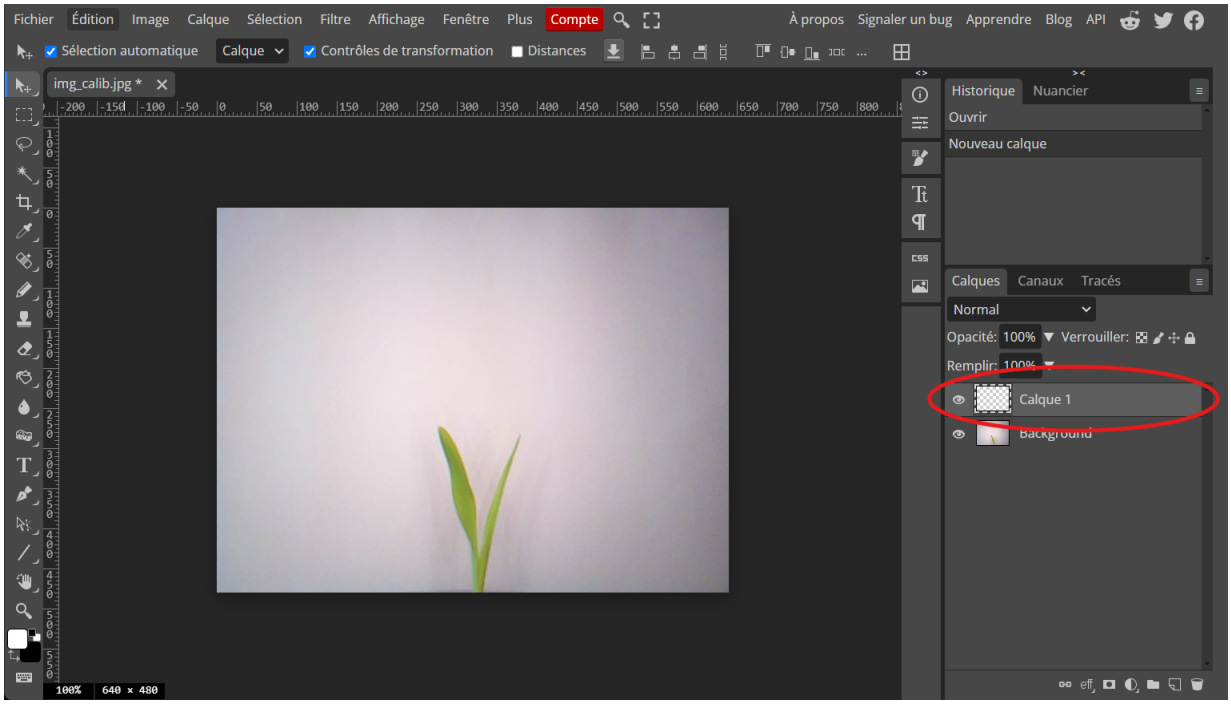


Figure F.3: Adding a layer

Create the reference skeleton

We will now create the reference skeleton on the newly added layer. To do this, we start by taking the *Pot de peinture* tool in the left bar. We select the black color in the background on the left and then we click on the image, making sure that it is the layer that is selected. To do this, simply check in the bar on the right if *Calque* is grayed out. Once this is done, we will set the opacity of the layer between 50 and 70%.

Now we will draw the skeleton itself. To do this, select the pencil tool ①. Next, we need to change the color in the bottom left corner to white ②. We also need to adjust the pencil thickness in the top bar to 3 pixels ③. Finally, we can draw the skeleton of the plant ④. We must try to do this as best as possible because this skeleton, in the hypotheses, is used as a truth.

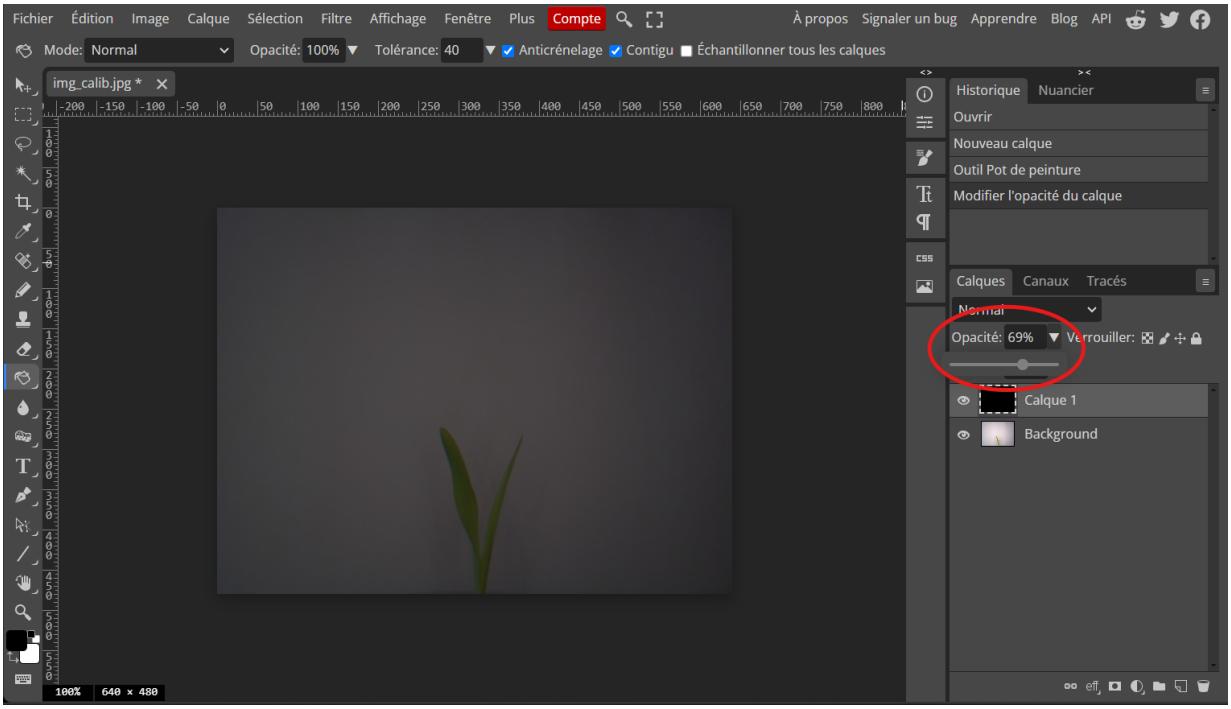


Figure F.4: Adjust the opacity of the layer

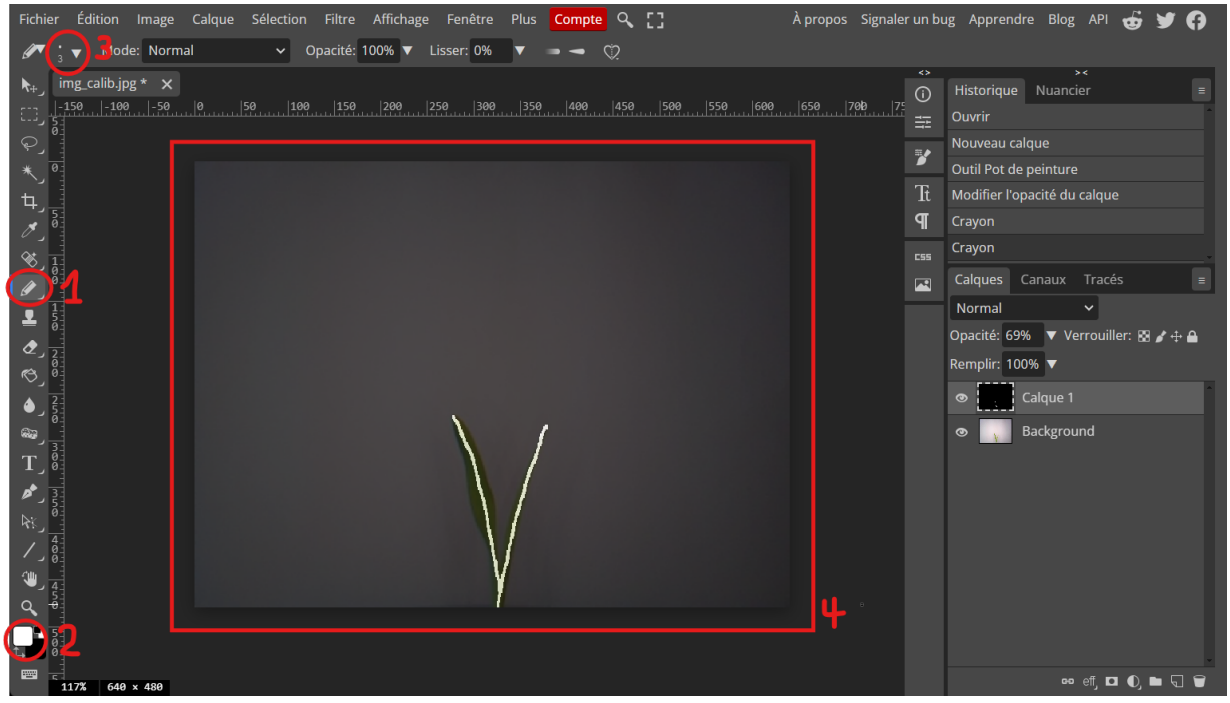


Figure F.5: Drawing of the skeleton

Exporting skelton_ref

Now we're at the final step. First, we need to reset the opacity to 100% in the same way as explained above. Next, all we have to do is export the image. To do this, go to *File* > *Export as* > *JPG*. ⚠ The JPG format is very important; if the image is saved in another format, the station will not recognize it. Finally, in the window that opens, we can specify the name of the image, here, *skeleton_ref* and click on *Enregistrer*.

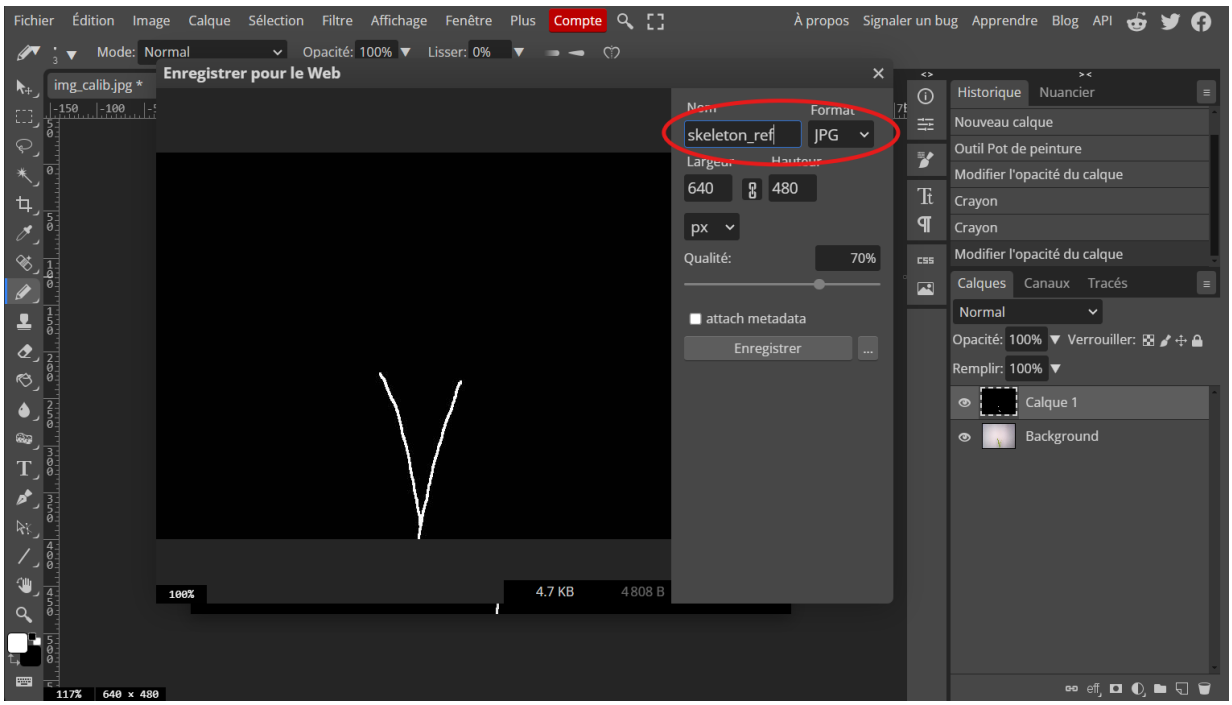


Figure F.6: Skeleton export

Appendix G

Use of artificial intelligence tools

In this dissertation, various artificial intelligence tools were used to facilitate certain stages of writing and development:

- **Translation:** Google Translate[1] and DeepL[2] were used to translate passages between English and French.
- **Documentary Research:** ChatGPT[3] and Gemini[4] provided support for directing readers to relevant sources.
- **Writing and Development Assistance:** ChatGPT was used to create and correct portions of computer code, as well as to proofread and reword certain passages of the text.

These tools were used in a complementary and critical manner, without replacing personal analysis, to improve the clarity and quality of the dissertation.

[1] https://translate.google.com/
[2] https://www.deepl.com/
[3] https://chatgpt.com/
[4] https://gemini.google.com/

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