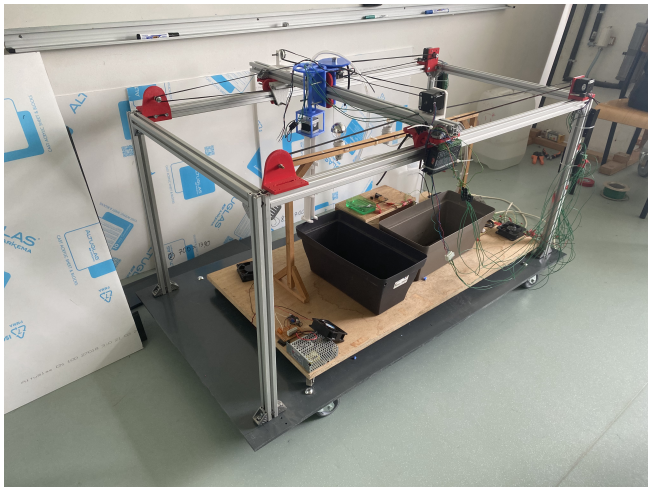

Connected Greenhouse - CDD

Axel MARTINS DE ARANJO
Ulrich PARÉ
Jérôme DUFEIL (first semester)
Engineering student, 4A GPSE

Raphaël CANALS
Teacher in Embedded Systems
Referant teacher

April 28, 2024

Summary

Acknowledgements

Firstly, we would like to thank our teachers for enabling us to acquire knowledge across the various domains necessary for the completion of this project. Next, we would like to express our gratitude specifically to Mr. Canals for being our supervising teacher throughout this annual project. Lastly, we appreciate Mr. Ladrouz for attentively addressing our component procurement needs, as well as assisting us in implementation; thanks to his support, we have overcome several hurdles in the process.

1. Project description

1.1. Analysis and expression of needs

The objective of this project is to establish a means of easily testing the optimal conditions for plant growth. This can vary depending on the context, but the aim may be to determine the parameters for the plant to grow fastest to a certain stage of its development, or to make it as voluminous, tall, colorful, etc., as possible. This will depend on the context and the user's needs. Additionally, the idea is to have the ability to remotely monitor the progress of this plant easily, only needing to occasionally refill the water reservoir.

This greenhouse should minimize human intervention on the plant as much as possible. Its objective is therefore to be used for plant research purposes. For example, there are giant connected greenhouses that autonomously manage the growth of lettuce and inform operators which plants are ready for harvest when they reach maximum yield. However, testing on such large greenhouses is quite complicated. In case of an inconclusive test that leads to the growth of mushrooms or any disease, the entire greenhouse would need to be cleaned, incurring significant costs and cleaning delays. It is in this context that our project makes sense.

The specifications require on-site control, achievable through the use of a touchscreen directly integrated into the greenhouse. Additionally, a dedicated software on a PC, connected via WiFi, offers a user-friendly interface for detailed and real-time management of various components of the greenhouse. For increased flexibility, remote access is provided through software on an online-connected PC, allowing users to monitor and regulate their greenhouse from anywhere.

1.2. Use case

The use case would be as following: Imagine an innovative connected greenhouse designed for growing medicinal plants. Using advanced technology, this greenhouse provides an optimal environment for the growth and flowering of medicinal plants, while allowing users to monitor and control every aspect of the cultivation process. We are seeking to develop our connected greenhouse in this context, both as a simpler example to understand and because our greenhouse is currently capable of accommodating it. Therefore, we will discuss later that our greenhouse aims to study the growth of salads.

- **Initial configuration :** The user configures the greenhouse by adding information relating to the medicinal plants to be grown. The integrated library offers pre-configured parameters for various medicinal plants, but the user can also add new plants with their own unique specifications.

- **Real-time monitoring :** The greenhouse displays crucial data such as temperature, humidity, brightness and watering cycles specific to each medicinal plant in real time. Intuitive icons visually represent each parameter, providing an instant view of each plant's status.

- **Watering automation :** The system automatically detects the water requirements of each medicinal plant based on built-in sensors. Watering is automatically activated according to the specific needs of each plant, ensuring precise and optimal hydration.

- **Precise Environmental Control:** The user manually adjusts parameters such as temperature, humidity and brightness to create ideal conditions for the growth stages of medicinal plants. Adjustment buttons offer intuitive handling, and changes made are visually confirmed on the screen.

- **Important Event Notification :** The greenhouse sends notifications in the event of exceptional conditions or specific needs detected for a given medicinal plant. Notifications can include manual adjustment advice or alerts concerning events such as impending flowering.

- **Harvesting and transfer :** When medicinal plants have reached maturity, the user receives a ready-to-harvest notification. Once harvested, plants can be transferred to their final location for medicinal use. By integrating these advanced features, the Connected Greenhouse offers an efficient and supervised means of growing medicinal plants, guaranteeing optimum yields while simplifying the process for users, whether health professionals or passionate amateurs.

1.3. Functional specifications

In this section, we will explore several diagrams and tables to set objectives and gather all our expectations for the final product in one place. First, we will examine a interaction graph along with a table summarizing all the functions associated with it. Next, we will have a FAST diagram (Function Analysis System Technique). Finally, we will have an FMECA (Failure Modes, Effects, and Criticality Analysis) to highlight the measures we need to implement to prevent our greenhouse from failing.

1.3.1. Interaction graph

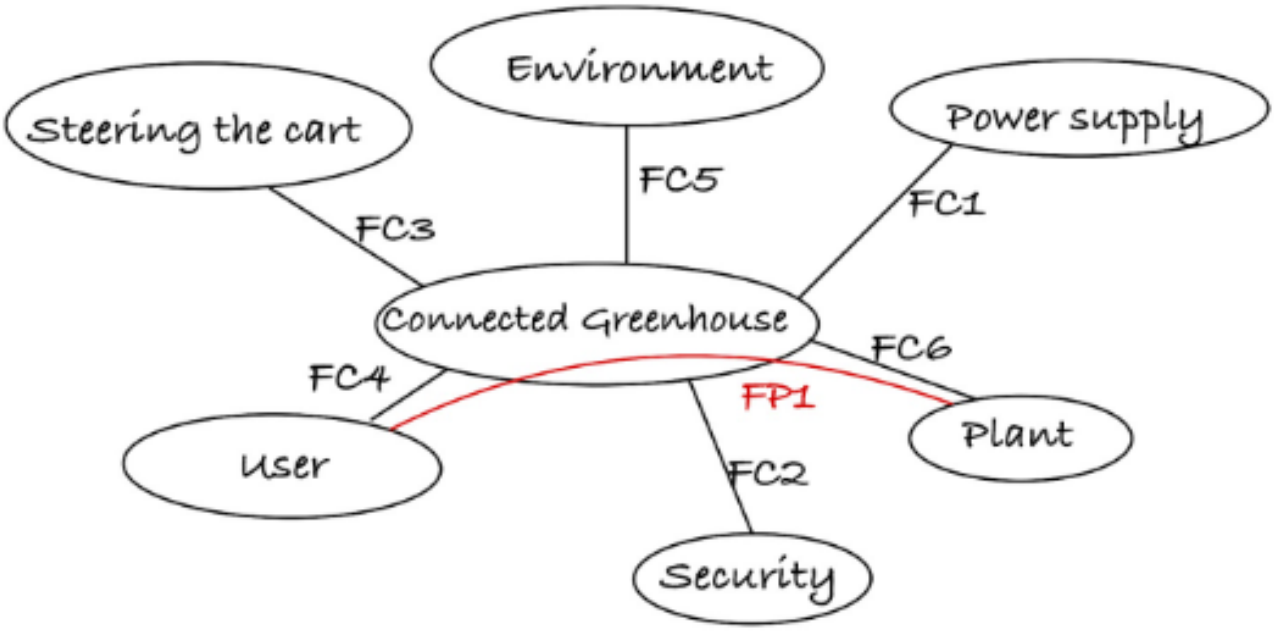

Figure 1: Interactions graph

Function	Requirements	Level
FP1 : Set all the greenhouse parameters so that the plant grows optimally	- Inform the user of the greenhouse status. - Allow users to control plant growth. - Automatically manage the physical parameters necessary for plant development.	- Display for each parameter and direct reading. - On a computer and/or a smartphone. - Refer to a database and control the hygrometry, humidity, temperature and luminosity.
FC1 : Ensure a power supply.	- Nature of energy. - Useful voltage. - Autonomy. - Contingency plan in the event of a power outage to maintain the proper functioning of the greenhouse.	- Electrical (Mains). - 230 V. - Continuous operation. - Inverter for essential components.
FC2 : Implement devices to ensure the integrity of the greenhouse.	- Installation of a sensor for the trolley's stepper motor, to prevent any movement limits from being exceeded. - Functional emergency stop.	- End of stroke sensor. - Emergency stop button.
FC3 : Control the position of the different elements of the greenhouse.	- Use of a manual joystick to control the motors for intuitive user interaction. - Implementation of safety devices to avoid accidents linked to the movement of engines. - Regular adjustment and maintenance of the mechanics to ensure smooth and precise operation of the engine.	- Different speed modes.
FC4 : Ensure user safety.	- Implementation of an emergency stop to instantly interrupt all operations in the event of a critical situation, thus ensuring the safety of users. - Accessibility to security devices. - Emergency signaling.	- Emergency stop button. -Easy access to emergency stop button. -Sound or light alarm.
FC5 : Ensure the safety of equipment and plants.	- Prevent weather problems such as humidity, rain, wind or high heat. - Integrate local weather data to anticipate climate variations and adjust instructions accordingly.	- Waterproofing and security of access. - Integration of external data.
FC6 : Adapt the instructions according to the plant.	- Have a system with specific information about each type of plant, including needs for temperature, humidity, light, watering frequency, etc.	- Plant database.

Table 1: Summary of interactions

1.3.2. F.A.S.T. (Function Analysis System Technique)

A Function Analysis System Technique (FAST) diagram is a powerful visual tool used in systems engineering to illustrate the relationships between functions and their contributing elements within a system. The primary purpose of a FAST diagram is to provide a structured framework for understanding how various functions are interconnected and how they contribute to achieving the overall objectives of the system. The key benefits of using a FAST diagram include:

Clarity and Visualization: By mapping out the relationships between functions and their components, a FAST diagram provides a clear and visual representation of the system's structure. This clarity helps stakeholders understand the system's architecture and how different elements interact with each other.

Identification of Critical Functions: FAST diagrams help identify critical functions within a system by highlighting their dependencies and relationships with other functions. This identification is crucial for prioritizing resources and focusing efforts on areas that have the most significant impact on system performance.

Problem Solving and Decision Making: FAST diagrams facilitate problem-solving and decision-making processes by providing a systematic approach to analyzing functions and their associated elements. By visualizing the flow of functions and their dependencies, stakeholders can identify potential issues, evaluate alternatives, and make informed decisions.
To create a FAST diagram effectively, follow these steps:

Identify Functions: Start by identifying the primary functions of the system. These functions represent the desired outcomes or objectives that the system is designed to achieve.

Decompose Functions: Break down each primary function into its constituent elements or sub-functions. These sub-functions represent the specific tasks or processes necessary to accomplish the primary function.

Establish Relationships: Determine the relationships between functions and their contributing elements. Use arrows or lines to denote the flow of influence or dependency between functions and their components.

Prioritize Functions: Assess the importance of each function based on its contribution to achieving the system's objectives. Prioritize critical functions that have a significant impact on system performance or functionality.

Review and Refine: Review the FAST diagram to ensure accuracy and completeness. Refine the diagram as needed to clarify relationships, improve readability, and address any discrepancies or omissions.
Overall, a well-constructed FAST diagram provides valuable insights into the structure and functioning of a system, enabling stakeholders to make informed decisions and effectively manage system complexity.

Main function	Sub-function	Solution	Criteria
FP1 : Inform the user of the greenhouse state	FP1.1 : Communicate the coordinates of the cart	Stepper-motors, microcontroller,HMI	Knowing the position with an accuracy of 3 millimeters
	FP1.2 : Communicate the quantity of water remaining in the reservoir	Auto-incrementation in software (with normalized refill)	Knowing the quantity of water with an accuracy of 10cl
	FP1.3 : Communicate all the parameters of the internal greenhouse of the greenhouse	HMI	
FP2 : Allow the user to control plant growth	FP2.1 : Master important elements of the internal environment for plant growth	Sensors, software, fans, LED spots etc.	
	FP2.2 : Control these elements at a distance	microcontroller (ESP-32)	
	FP2.3 : Check these elements on site	Touchscreen	
FP3 : Automatically manage the physical parameters	FP3.1 : Follow indications given by the soft	Data-base	
	FP3.2 : Applicate indications	Fans, LED spots, watering etc.	
FP4 : Visualizing the plant(s)	FP4.1 : Driving the cart that accommodates the camera	Stepper-motors, microcontroller, mecanicals parts	
	FP4.2 : Devices to look at the plant	Camera (ESP-cam)	
FP5 : Ensure power to the greenhouse as a whole	FP5.1 : Adapt the current and the tension for each of the components	Hardware	
	FP5.2 : Convert the Electrical Network for a Signal suitable for our greenhouse	Transformer	
FP6 : Set up devices ensuring the integrity of the greenhouse and the user	FP6.1 : Detect when teh greenhouse is at the end stop	End-of-travel sensor	
	FP6.2 : Emergency stop	Emergency button	
FP7 : Allowing every internals parameters of the greenhouse	FP7.1 : Knowing the humidity of the air	SHT31-D	±3%
	FP7.2 : Knowing the humidity of the dirt	Adafruit STEMMA	±3%
	FP7.3 : Knowing the internal luminosity	TSL2691	±50lux
	FP7.4 : Knowing the internal temperature	SHT31-D	±2°C

Table 2: F.A.S.T Diagram

1.3.3. F.M.E.C.A. (Failure Modes, Effects, and Criticality Analysis)

In the context of Failure Modes, Effects, and Criticality Analysis (FMECA), determining the criticality of a function involves assessing the severity of potential failures based on three key factors: detectability, occurrence, and severity.

- **Detectability (D):** This factor evaluates how easily a failure mode can be detected before it leads to adverse effects. A high detectability rating means that the failure mode is easily identifiable, whereas a low rating indicates that it may go unnoticed until it causes significant problems.

- **Occurrence (O):** It refers to the frequency or likelihood of a failure mode happening during the operation of the system. A high occurrence rating means that the failure mode is likely to happen frequently, while a low rating suggests that it occurs infrequently.

- **Severity (S):** It assesses the impact or consequences of a failure mode on the system's performance, safety, or functionality. A high severity rating implies that the failure mode would have severe consequences, whereas a low rating indicates that the consequences are minimal.

To calculate the **criticality (C)** of a function, you multiply the ratings for detectability, occurrence, and severity together. This approach helps prioritize which failure modes require immediate attention and mitigation efforts. Functions with higher criticality scores indicate that they pose a greater risk to the system's performance, safety, or reliability and should be addressed with appropriate preventive or corrective actions.

Product function	Failure mode	Effect of failure	Cause of failure	Rating (1 to 4)				Solutions to put in place
				D	O	S	C	
FP1.1 : Communicate the coordinates of the cart	Loss of communication with the cart	Inability to track the cart's location	Signal interference or hardware malfunction	1	2	2	4	Implement redundant communication channels, use error-checking protocols, regularly maintain communication hardware.
FP1.2 : Communicate the quantity of water remaining in the reservoir	Incorrect water level indication	Over or underwatering of plant	s Sensor malfunction or calibration error	1	3	3	9	Regularly calibrate sensors, install backup sensors, implement automated alerts for irregularities.
FP1.3 : Communicate all the parameters of the internal greenhouse of the greenhouse	Inaccurate data transmission	Incorrect environmental control decisions	Communication network failure or sensor	1	2	2	4	Implement redundancy in communication systems, conduct regular sensor maintenance, perform system checks.
FP2.1 : Master important elements of the internal environment for plant growth	Incomplete monitoring of environmental factors	Inadequate plant growth conditions	Sensor malfunction or inadequate sensor coverage	2	2	2	8	Expand sensor coverage, implement redundant sensors, conduct regular calibration checks.
FP2.2 : Control these elements at a distance	Loss of remote control functionality	Inability to adjust environmental conditions remotely	Communication network failure or control system malfunction	1	3	2	6	Redundant control systems, ensure robust communication networks, conduct regular system checks.
FP2.3 : Check these elements on site	Inability to perform on-site checks	Delayed or missed detection of environmental irregularities	Sensor malfunction or limited access to greenhouse	1	3	2	6	Automated monitoring systems, conduct regular sensor maintenance, establish protocols for manual checks.
FP3.1 : Follow indications given by the soft	Software malfunction or incorrect instructions	Incorrect environmental adjustments	Software bugs or user error	1	3	2	6	Regular software updates and maintenance, user training, implement error-checking protocols.
FP3.2 : Applicate indications	Inability to execute software instructions	Failure to implement required environmental adjustments	Control system malfunction or user error	2	3	2	12	Regular maintenance of control systems, user training on system operation, implement error-checking protocols.
FP4.1 : Driving the cart that accommodates the camera	Cart movement failure	Inability to adjust camera position	Motor malfunction or control system failure	2	3	3	18	Regular maintenance of cart mechanisms, implement redundant motor systems, conduct regular system checks.

FP4.2 : Devices to look at the plant	Camera malfunction or obstruction	Inability to monitor plant growth	Camera hardware failure or environmental obstruction	2	3	2	12	Regular maintenance of camera systems, implement backup camera systems, ensure clear camera visibility.
FP5.1 : Adapt the current and the tension for each of the components	Incorrect voltage or current supply	Damage to electronic components	Power supply malfunction or incorrect settings	2	3	2	12	Regular maintenance of power supply systems, implement voltage regulation systems, conduct regular system checks.
FP5.2 : Convert the Electrical Network for a Signal suitable for our greenhouse	Signal conversion failure	Incompatibility with greenhouse systems	Incompatibility with greenhouse systems	1	3	2	6	Regular maintenance of signal conversion systems, implement redundant conversion systems, ensure compatibility testing.
FP6.1 : Detect when the greenhouse is at the end stop	Failure to detect cart position	Risk of collision or damage	Sensor malfunction or calibration error	2	3	4	24	Regular maintenance of position detection systems, implement redundant sensors, conduct regular calibration checks.
FP6.2 : Emergency stop	Inability to activate emergency stop	Inability to halt greenhouse operations in emergencies	Control system malfunction or user error	2	3	4	24	Regular maintenance of emergency stop systems, user training on emergency procedures, implement backup emergency systems.
FP7.1 : Knowing the humidity of the air	Incorrect humidity measurement	Inaccurate environmental control decisions	Sensor malfunction or calibration error	1	2	3	6	Regular calibration of humidity sensors, implement redundant sensors, conduct regular system checks.
FP7.2 : Knowing the humidity of the dirt	Incorrect soil moisture measurement	Over or underwatering of plants	Sensor malfunction or calibration error	1	2	3	6	Regular calibration of soil moisture sensors, implement redundant sensors, conduct regular system checks.
FP7.3 : Knowing the internal luminosity	Incorrect light intensity measurement	Inadequate lighting conditions for plant growth	Sensor malfunction or calibration error	1	2	3	6	Regular calibration of temperature sensors, implement redundant sensors, conduct regular system checks.
FP7.4 : Knowing the internal temperature	Incorrect temperature measurement	Inaccurate environmental control decisions	Sensor malfunction or calibration error	1	2	3	6	Regular calibration of temperature sensors, implement redundant sensors, conduct regular system checks.
FP7.5 : Knowing the external temperature	Incorrect temperature measurement	Inaccurate environmental control decisions	Sensor malfunction or calibration error	1	2	3	6	Regular calibration of temperature sensors, implement redundant sensors, conduct regular system checks.

Table 3: F.M.E.C.A.

2. System Architecture

After establishing our various functions and briefly outlining the different components we will use, we will delve into these components in more detail in this section, as well as the broad outlines of the development of this connected greenhouse. Below, we can see a diagram representing the architecture of our system.

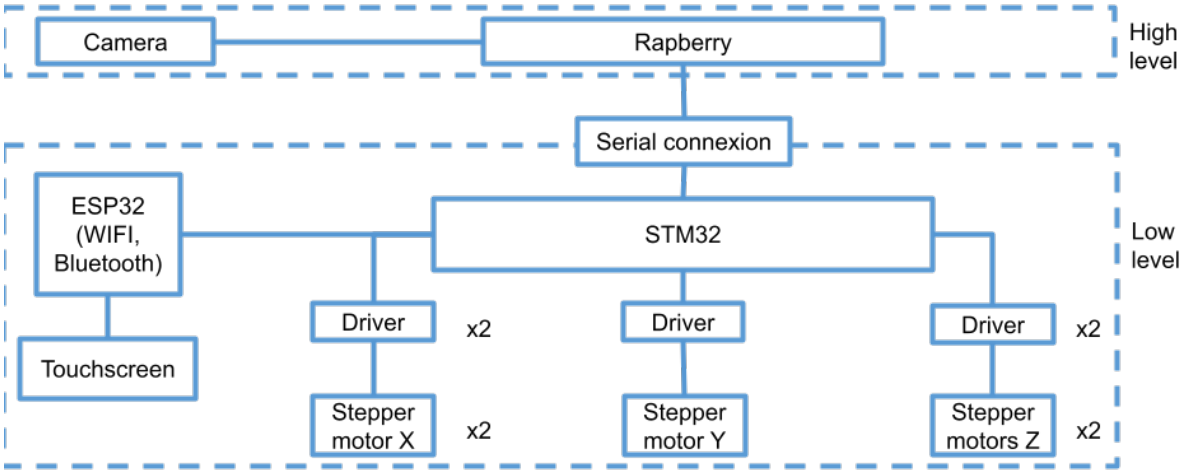

Figure 2: Schematic showing system architecture

2.1. Mechanical Design

2.1.1. Introduction

To begin with, one of the main issues we noticed upon taking over this project was the lack of mechanical stability in the components, their insufficient robustness, and imprecise guidance of the X and Y axes due to excessive play and a need for reevaluation in sizing. This is why this aspect, which may seem out of place in the project, is indeed emphasized.

2.1.2. Objectives and goals

- Addressing the lack of stability and imprecision in the guidance
- Reinforcing the parts that have broken and not replicating them as they were
- Validating the new design for testing in 3D printing
- Conducting tests under real-world conditions with the new components
- Manufacturing these parts with stronger materials if previous tests have positive results

2.1.3. Some CAD pictures

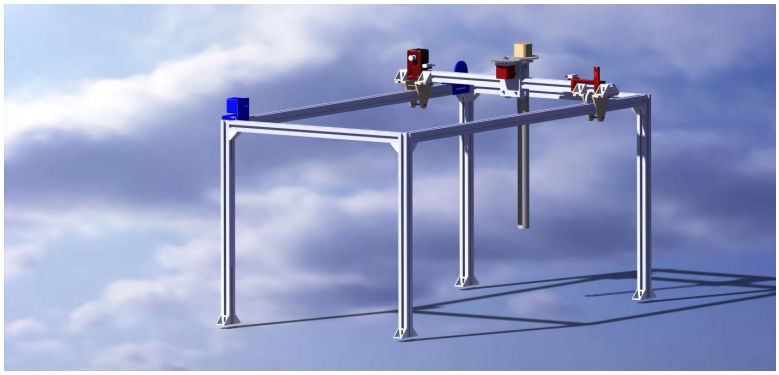

Figure 3: Full assembly

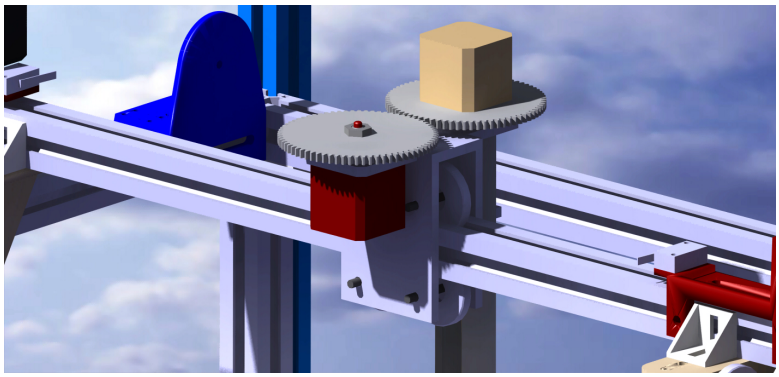

Figure 4: Z-transmission assembly

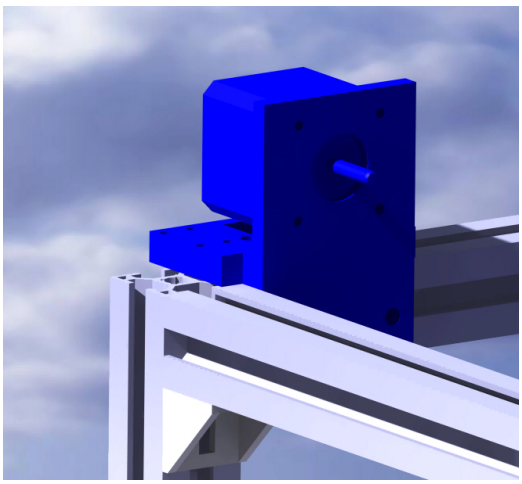

Figure 5: X-Motor

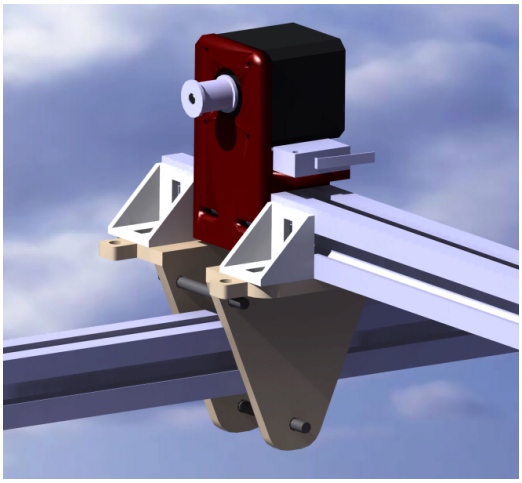

Figure 6: Y-Motor

2.2. Greenhouse automation

2.2.1. Introduction

The main objective of the mission is to implement a motor control system in the connected greenhouse, aimed at efficiently automating various processes associated with the motors. This automation will help optimize farming operations, improve productivity, and ensure precise control of motorized equipment in the greenhouse environment.

2.2.2. Objectives and goals

- Set up a motor control system in the connected greenhouse to effectively automate various processes associated with motors.
- Optimize productivity and ensure precise control of motorized equipment in the greenhouse.
- Automation of motorized tasks: Adjustment of fans, control of irrigation systems
- Connectivity and remote control: monitor and adjust operating parameters via a web interface.
- Responsiveness to environmental conditions: Sensors.

2.2.3. Components list

To optimize the growth of lettuce in our greenhouse, we've selected specific components based on their functionality, efficiency, and compatibility with the environmental needs of lettuce cultivation. Here's a detailed explanation of why each component was chosen:

Fans :

- Specifications: 5V DC, 35mm x 35mm x 10mm, 680mW power consumption, 7.2cfm airflow.
- Rationale: The chosen fans are crucial for maintaining air circulation, which is essential to prevent disease and provide a stable environment. The size and power of these fans are ideal for our greenhouse space, ensuring adequate airflow without being overly disruptive.

Figure 7: Fan

Moisture Sensor (Adafruit STEMMA Soil Sensor):

- Specifications: Capacitive moisture measurement, readings from 200 (very dry) to 2000 (very wet), temperature sensor included.
- Rationale: The capacitive measurement technology avoids corrosion and electrical interference with the plant's growth environment. This sensor helps monitor and maintain optimal soil moisture levels essential for lettuce, which thrives in slightly moist conditions. The Soil temperature must be between 6 and 16°C for good yields. Ideal soil moisture is generally between 60% and 70% of the soil's water retention capacity.

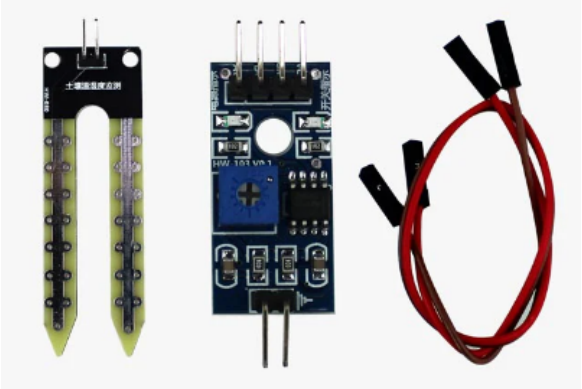

Figure 8: Adafruit STEMMA Soil Sensor

Air Temperature and Humidity Sensor (Adafruit SHT31-D):

- Specifications: ±2% relative humidity and ±0.3°C temperature accuracy.
- Rationale: Lettuce requires a cool environment with specific humidity levels to thrive (between 10°C and 20°C during the day and no less than 5°C at night). This sensor's high accuracy ensures we can closely monitor and maintain the ideal air temperature and humidity, crucial during the germination and vegetative growth phases.

Figure 9: Adafruit SHT31-D

Light Sensor (Adafruit TSL2591):

- Specifications: Dynamic range from 1 to 600,000,000 counts, measures up to 88,000 Lux.
- Rationale: Light intensity is critical for photosynthesis and overall plant health. This sensor enables us to accurately measure and adjust our lighting system to provide optimal light levels throughout the plant's lifecycle.

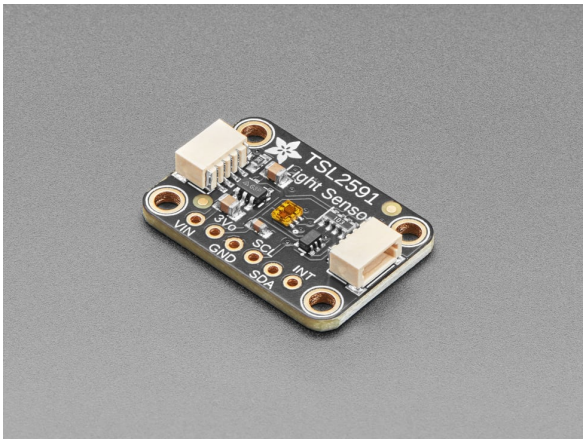

Figure 10: Adafruit TSL2591

Relay (Ningbo Songle Relay SRD-5VDC-SL-C):

- Specifications: 5V DC coil voltage, capable of handling up to 10A.
- Rationale: The relay acts as an intermediary to safely control the power to our higher voltage systems like fans and lights, ensuring they operate only when necessary to maintain environmental conditions.

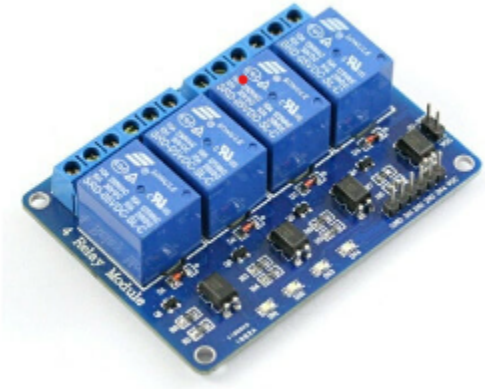

Figure 11: Ningbo Songle Relay SRD-5VDC-SL-C

Lamps (Lexman SCL G45AL E14 H):

- Specifications: 25W, 250 lumens per bulb, 2700K color temperature.
- Rationale: These lamps provide a warm white light suitable for promoting growth. The light intensity and spectrum are especially beneficial during the seedling and vegetative stages of lettuce, facilitating healthy growth and development.

Figure 12: Lamps

Raspberry Pi4 8GB:

- Specifications:

SoC	Broadcom BCM2711
CPU	64-bit ARM Cortex-A72 (4x 1.5 GHz)
GPU	Broadcom VideoCore VI
RAM	Up to 8 GB LPDDR4

Table 4: Raspberry Pi4 specifications

Figure 13: Raspberry Pi4 8GB

Each component has been selected not only for its individual capabilities but also for its synergy with other elements of the system. This ensures a balanced, controlled, and efficient growth environment, tailored to the specific needs of lettuce cultivation in our greenhouse.

2.2.4. Pin attribution

Component	Label	Pin	Use
Moisture sensor 1	sensorPin	33	Analogic reading of moisture for the first sensor
Moisture sensor 2	sensorPin2	32	Analogic reading of moisture for the second sensor
Relay of the pump	pumpRelayPin	16	Pump start to water plant
Relay of lights	relayPinLights	17	Switch on the lights
Relay of fans	relayPinFans	14	Start to turn the fans
X-axis Stepper motor 1	stepPinX1	4	Give the speedrate of stepper motor X1
X-axis Stepper motor 1	dirPinX1	2	Give the direction of stepper motor X1
X-axis Stepper motor 2	stepPinX2	13	Give the speedrate of stepper motor X2
X-axis Stepper motor 2	dirPinX2	12	Give the direction of stepper motor X2
X-axis Stepper motors	enablePinX	5	Give the authorisation to X-axis stepper motors to turn
Y-axis Stepper motor	stepPinY	19	Give the speedrate of stepper motor Y
Y-axis Stepper motor	dirPinY	18	Give the direction of stepper motor Y
Y-axis Stepper motor	enablePinY	15	Give the authorisation to Y-axis stepper motor to turn
Z-axis Stepper motor	stepPinZ	26	Give the speedrate of stepper motor Y
Z-axis Stepper motor	dirPinZ	25	Give the direction of stepper motor Y
Z-axis Stepper motor	enablePinZ	27	Give the authorisation to Z-axis stepper motor to turn

Table 5: Pins allocation table

This table show every pins used in the C code televersed in the ESP32. It regroups the component which use the pin, how is it named in the code, the number of the pin concerned and for what function is it use.

NB : for the stepper-motors, it isn't connected directly to the motors but through the driver which control the motor.

2.2.5. Existing functions (C code for ESP32)

Function	Use
void moveX(**int** stepPin1, **int** stepPin2, **float** distance, **bool** forward, **int** speed)	Turn the X-axis stepper motors for given distance with a speed (turn/min) and in a given direction (TRUE = positive, FALSE = negative). stepPin refers to Table 4
void moveY(**int** stepPin, **float** distance, **bool** forward, **int** speed)	Turn the Y-axis stepper motor for given distance with a speed (turn/min) and in a given direction (TRUE = positive, FALSE = negative). stepPin refers to Table 4
void moveZ(**int** stepPin, **float** distance, **bool** forward, **int** speed)	Turn the Z-axis stepper motor for given distance with a speed (turn/min) and in a given direction (TRUE = positive, FALSE = negative). stepPin refers to Table 4
void simpleMoveX(**int** stepPin1, **int** stepPin2, **int** steps)	Turn the X-axis stepper motors to test other function easily. stepPin refers to Table 4
void simpleMoveY(**int** stepPin, **int** steps)	Turn the Y-axis stepper motor to test other function easily. stepPin refers to Table 4
void simpleMoveZ(**int** stepPin, **int** steps)	Turn the Z-axis stepper motor to test other function easily. stepPin refers to Table 4
void temperatureFanControl()	Function to turn fans on when the temperature sensors measure a above threshold measure
void lightControl()	Function to switch lights on when the light sensor measure a below threshold measure

Table 6: Existing functions (C)

2.2.6. Existing functions (Python code for Raspberry HMI)

Function	Use
move_x(direction)	Turn the X-axis stepper motors for one step (1.8°) for a given direction. This time, direction is a string "forward" or "backward" respectively meaning positively and negatively
move_y(direction)	Turn the Y-axis stepper motor for one step (1.8°) for a given direction. This time, direction is a string "forward" or "backward" respectively meaning positively and negatively
move_z(direction)	Turn the Z-axis stepper motor for one step (1.8°) for a given direction. This time, direction is a string "up" or "down" respectively meaning positively and negatively
r0()	Execute move_x, move_y, move_z functions in a specific order to set X, Y and Z to 0.
goTo(X_coord, Y_coord, Z_coord)	Execute move_x, move_y, move_z functions in a specific order to set X, Y and Z to respectively X_coord, Y_coord and Z_coord.
save_coords()	Save coordinates that user wants to set the trolley position. Theses values are written in tk.entry object of the HMI.
showPosition()	Show the position of the trolley in the console
speedRate(s)	Set the speedrate of the trolley in turn/min. "s" is a level going from 1 to 3. When s increase, the speedrate increase too.
open_data_display_window()	Function to display the data measured by sensors
open_motor_control_window()	Function to display the control of motors

Table 7: Existing functions (Python)

3. Environmental and Social Responsibility approach

Reducing electricity consumption in standby state is a crucial goal in today's energy-conscious world. Standby power, also known as vampire power or phantom load, refers to the energy consumed by electronic devices when they are switched off but still plugged in. This standby power consumption may seem insignificant for individual devices, but collectively, it amounts to a substantial waste of energy and money.

One effective way to address this issue is by implementing advanced power management techniques. Devices can be designed to enter a low-power or sleep mode when not in use, significantly reducing their standby power consumption. Additionally, smart power strips can be utilized to automatically cut off power to devices that are not in use, preventing them from drawing unnecessary electricity.

Another approach is to utilize energy-efficient components and technologies in the design of electronic devices. For example, using high-efficiency power supplies and components with low standby power consumption can help minimize energy waste. Additionally, incorporating energy-saving features such as automatic power-off timers and motion sensors can further reduce standby power consumption.

In the context of greenhouse construction, selecting the best materials for the plates to close the structure is crucial for optimizing energy efficiency. The plates play a significant role in maintaining the internal environment of the greenhouse by providing insulation and controlling heat exchange with the external environment. Materials with excellent thermal insulation properties, such as double-glazed glass or polycarbonate panels, can help minimize heat loss during colder months and reduce the need for heating.

Furthermore, proper insulation of the internal environment is essential for efficiently managing temperature variations within the greenhouse. Insulating materials, such as foam insulation or reflective barriers, can be used to create a thermal barrier that prevents heat transfer between the interior and exterior environments. This insulation not only helps maintain a more stable internal temperature but also reduces the energy required to heat or cool the greenhouse as needed.

In conclusion, addressing standby power consumption and optimizing thermal insulation are critical steps in promoting energy efficiency and sustainability. By employing advanced power management techniques and selecting appropriate materials for greenhouse construction, we can significantly reduce energy waste and minimize our environmental footprint.

List of Figures

List of Tables

Bibliography

Sites :
https://www.raspberrypi.com/documentation/accessories/display.html
https://www.makerguides.com/tb6600-stepper-motor-driver-arduino-tutorial/
https://www.redohm.fr/2016/04/hardward-driver-moteur-pas-a-pas/

Youtube :
https://www.youtube.com/watch?v=gREPSNzmfWI
https://www.youtube.com/watch?v=sIym-Iz99P0
https://www.youtube.com/watch?v=SUMYJJ6ulaM
https://youtu.be/TQ7R2bY-MWU?si=MuG-oTX5hqTod6-J
https://youtu.be/fHAO7SW-SZI?si=6ZSu0amqpOVnbg2m
https://youtu.be/zUb8tiFCwmk?si=R4HgoRyb6-TXX0SH
https://youtu.be/nLV0fjUWI-g?si=I597rR514-J5CBhA
https://www.youtube.com/watch?v=LGFNh-JwpEk
https://youtu.be/Yvth5tLuXd4?si=ZoKFlJP4EJbU99SH
https://youtu.be/idVcItHfGS4?si=UdE6Zq0u75TuFDt0
https://youtu.be/QRCvC5xhJCw?si=1wlyuEtb2XpbPzaf
https://youtu.be/Xlkmso01vUk?si=qQOdnvpRkPqmm8
https://youtu.be/Xz4iVpdMd-w?si=QDLdWVWeC0hhmAB
https://youtu.be/npLGj4aBkFo?si=mlVVqwKdCv3ekC5s
https://youtu.be/EWbOMhBc-Us?si=mFd5gGH-HZc-jIua

Playlist Youtube :
https://www.youtube.com/watch?v=sW-5IewOn-g&list=PLb1SYTph-GZJOhtl6fwqc4wDZfPL2DalE&index=1

Pinterest :
837dd46490d9234975ffe4a3bb4f68d7.jpg
88a5f446cf30486786f6f01fcffa41e1.jpg
55c873256cc9c0c21a0b8185c2f02276.jpg
b8848dff0c9a939d32a798bfb70f15ef.jpg
150a26db6f0cbbe1b18210e3b3c3f5b2.jpg
69288cd30025653530b1a9e4e132c6c0.jpg
fce1052b28debcb03f536387e10213c8.jpg
d1335a6ab4d2804144fd6aba649c0120.jpg

Axel MARTINS DE ARANJO felt

I was part of the team that worked on the Computer Assisted Design (CAD) and implementation of an Human-Machine Inteface (HMI) of the greenhouse. I also charged to do purchases lists for every components. As a member of the team, I was responsible for the following tasks:

- CAD : Thanks to my skills already acquired in the field of mechanical design through my previous training at the Institute of Technology of the University of Poitiers in the Mechanical Engineering and Production department, I was the ideal person among the group members to take care of the mechanical design of the greenhouse that needed to be revised. I didn't particularly improve my skills in this area, but I was able to reinforce my foundations by revisiting everything I had covered in previous years.
- HMI : With my colleague Ulrich, both of us worked on the human-machine interface of the greenhouse. As previously mentioned, this interface is accessible from a touchscreen. My task was to develop this interface with Ulrich and then incorporate it into the touchscreen.
- Project management: I was sort of the project manager. I took care of updating our Gantt chart on a weekly basis, and I also handled the ordering and tracking of components with Kamel Ladrouz that we needed, all members of the group and myself. I made significant progress in this area as I was able to learn about the utility of new components.
- Communication: I was able to improve my communication with the group members, especially by taking on the role of project manager, as I stayed informed about everyone's progress while also helping them with their tasks whenever I had some spare time. Additionally, I took care of sending emails or scheduling appointments with our supervising professor when it was necessary to meet due to development issues. I also handled the writing of this document almost entirely.
- 3D-Print :Before this project, I had no knowledge about 3D printing. I had never printed anything in my life, and it was a very interesting discovery, especially when researching the printing materials we would use.

Overall, I believe that my contribution to the design of the greenhouse has been significant, and I am proud of the final product. I believe that my experience in this project will be valuable in future projects, both in terms of technical skills and experience in project management.

Axel	
CAD	Intermediate → Intermediate+
HMI	Beginner → Intermediate
Project management	Intermediate → Intermediate+
Communication	Beginner→ Intermediate+
3D-Print	0 → Beginner+
LaTeX	0 → Intermediate

Ulrich PARÉ felt

As part of a project to design a connected greenhouse, I developed an on- board system to control the motors, as well as a complete environmental control system for the greenhouse. The system includes automatic watering, lighting and ventilation using fans, as well as control of ambient temperature and soil humidity using dedicated sensors. At the same time, I started developing an implementation of an on-site HMI in order to obtain real-time feedback on the state of the greenhouse and to ensure control of the cart motors.

- **Motor control :** Motor servoing in this project consists in precisely controlling the movements of the carriage inside the greenhouse. To achieve this, I used a total of 5 stepper motors. Of these, two motors are dedicated to the X axis, another to the Y axis, and two motors are used for the Z axis. These two motors for the z-axis enable both translational and rotational movements of the carriage on this axis for various purposes. Each stepper motor is controlled by a specific driver, enabling them to be servo-controlled using a microcontroller. The servo-control of the motors enables precise control of the cart's movements, guaranteeing efficient management of its positioning inside the greenhouse. Their use in conjunction with drivers offers several advantages, including precise control of positioning, increased repeatability of movements, and reduced energy consumption when stationary. These features are essential to ensure optimal operation of the connected greenhouse system.
- **Greenhouse environmental control :**Environmental control of the greenhouse is a crucial component of this project. To ensure optimal conditions for plant growth, I set up a complete system including:
  - **Automatic watering :**An automated system to ensure regular watering of plants according to their water requirements so an adequate level of moisture in the soil is maintained, thus promoting their growth.
  - **Lighting management :**A greenhouse lighting control system that simulates the ideal light conditions for plant growth by a lighting cycle according to specific crop needs.
  - **Ambient temperature control :**A dedicated sensor is used to monitor the temperature inside the greenhouse in real time. This sensor automatically regulates thermal conditions, activating or deactivating heating or cooling systems using fans.
  - **Soil moisture control :**A soil moisture sensor is integrated into the system to continuously monitor soil moisture levels. This enables watering cycles to be precisely adjusted to the specific needs of the plants, avoiding any risk of under- or over-watering.
- **Human Machine Interface (HMI):** For efficient greenhouse management, I've set up two human-machine interfaces (HMI):
  - **On-site HMI :** Together with Axel, we developed a user interface for local control of the greenhouse. This HMI allows users to monitor the greenhouse39;s environmental conditions in real time thanks to feedback on the status of the sensors and the ability to control the motors.

– **Remote HMI :** Although a remote HMI, this part of the project was not finalized as part of this development phase. However, initial development has begun, and will be continued in later phases of the project. The implementation of these user interfaces enables users to control and monitor the greenhouse intuitively and efficiently, providing a convenient way of managing growing operations remotely or on-site.

Ulrich	
Software (VSCode, Arduino)	Beginner → Expert
HMI	Beginner → Intermediate
Electronics skills	Intermediate → Expert
Java and HTML	Beginner → Intermediate
Project management	Intermediate → Intermediate+

Jérôme DUFEIL felt

I was a member of the team entrusted with the development and implementation of the database within the greenhouse project. Within this capacity, my responsibilities encompassed the following tasks:

- **Database Management:** Leveraging my existing SQL knowledge, I undertook the creation of a mySQL database using MARIADB. However, I encountered challenges in optimizing its efficiency to prevent unnecessary space consumption on the SD card integrated into the Nvidia Jetson Nano system. Regrettably, I also encountered technical issues during the installation of Node-RED which was intended to code the interactions between various components and the database. As a result, I couldn't improve my skills in this area.
- **CAPELLA:** Prior to this endeavour, my familiarity with CAPELLA was limited. Nonetheless, I assumed the responsibility of implementing our functional and technical specifications within this software framework. Despite making significant strides in comprehension, I concede that certain aspects of CAPELLA remain enigmatic to me, indicating that mastery has yet to be achieved.
- **Communication:** Over the course of the project, I made notable enhancements to my communication prowess. Collaborating closely with colleagues, we all engaged in deliberations to inform decision-making processes. Furthermore, my investment in oral presentations and the delivery of progress reports afforded me opportunities to refine my communication skills.

Ultimately, I believe I have made significant contributions to the project's research aspect. However, progress in the database segment has not advanced sufficiently to warrant discussion of substantial advancements. Nonetheless, I take pride in the work accomplished and have gained valuable project management skills throughout the process.

Jérôme	
MARIADB	0 → Intermediate
Project management	Beginner → Intermediate-
Communication	Beginner → Intermediate+
Node-Red	0 → Beginner

C code for ESP32 :

```
// Include necessary libraries
#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_SHT31.h"
#include "Adafruit_TSL2591.h"

// Define pins and variables for the first sensor
const int sensorPin = 33;        // Pin 33 used as ADC input
const int maxValue = 2240;       // Maximum sensor value representing 100%

// Define pins and variables for the second sensor
const int sensorPin2 = 32;       // Pin for the second sensor
const int maxValue2 = 2240;      // Maximum sensor value for the second sensor

// Define pin connected to the relay controlling the pump
const int pumpRelayPin = 16;

// Pins for X-axis
const int stepPinX1 = 4;
const int stepPinX2 = 13;
const int dirPinX1 = 2;
const int dirPinX2 = 12;
const int enablePinX = 5;

// Pins for Y-axis
const int stepPinY = 19;
const int dirPinY = 18;
const int enablePinY = 15;

// Pins for Z-axis
const int stepPinZ = 26;
const int dirPinZ = 25;
const int enablePinZ = 27;

// Number of rotations for each axis for sensor 1 and sensor 2
int numRotationsSensor1_X = 10;
int numRotationsSensor1_Y = 10;
int numRotationsSensor1_Z = 10;

int numRotationsSensor2_X = 15;
int numRotationsSensor2_Y = 15;
int numRotationsSensor2_Z = 15;

int state = 0; // State variable for state machine

Adafruit_SHT31 sht31 = Adafruit_SHT31();
Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591); // Creating an instance of the TSL2591 sensor

// Define relay pins for controlling lights and fans
const int relayPinLights = 17; // Pin connected to the relay controlling the lights
const int relayPinFans = 14;   // Pin connected to the relays controlling the fans

void setup() {
  // Initialize serial communication
  Serial.begin(115200);
  Serial.println("Moisture Sensor and Water Pump Control Example");

  // Set pump relay pin as output
  pinMode(pumpRelayPin, OUTPUT);
```

```
60
61    // Initialize pin modes for X-axis
62    pinMode(stepPinX1, OUTPUT);
63    pinMode(stepPinX2, OUTPUT);
64    pinMode(dirPinX1, OUTPUT);
65    pinMode(dirPinX2, OUTPUT);
66    pinMode(enablePinX, OUTPUT);
67
68    // Initialize pin modes for Y-axis
69    pinMode(stepPinY, OUTPUT);
70    pinMode(dirPinY, OUTPUT);
71    pinMode(enablePinY, OUTPUT);
72
73    // Initialize pin modes for Z-axis
74    pinMode(stepPinZ, OUTPUT);
75    pinMode(dirPinZ, OUTPUT);
76    pinMode(enablePinZ, OUTPUT);
77
78    // Initialize relay pins for lights and fans
79    pinMode(relayPinLights, OUTPUT);
80    pinMode(relayPinFans, OUTPUT);
81
82    // Initialize I2C communication
83    Wire.begin();
84
85    // Initialize SHT31 sensor
86    if (!sht31.begin(0x44)) {   // Set to 0x45 for alternate i2c addr
87      Serial.println("Couldn't find SHT31");
88      while (1) delay(1);
89    }
90
91    // Initialize TSL2591 sensor
92    if (!tsl.begin()) {
93      Serial.println("Couldn't find TSL2591 sensor");
94      while (1);
95    }
96
97    tsl.setGain(TSL2591_GAIN_LOW); // Set sensor gain to low
98    tsl.setTiming(TSL2591_INTEGRATIONTIME_100MS); // Set integration time to 100ms
99  }
100
101 void loop() {
102   // Read moisture level from the first sensor
103   int sensorValue1 = analogRead(sensorPin);
104   float percentage1 = (sensorValue1 / (float)maxValue) * 100.0;
105
106   // Read moisture level from the second sensor
107   int sensorValue2 = analogRead(sensorPin2);
108   float percentage2 = (sensorValue2 / (float)maxValue2) * 100.0;
109
110   // Print moisture levels for both sensors
111   Serial.print("Moisture Percentage (Sensor 1): ");
112   Serial.print(percentage1, 2);
113   Serial.println("%");
114
115   Serial.print("Moisture Percentage (Sensor 2): ");
116   Serial.print(percentage2, 2);
117   Serial.println("%");
118
119   // State machine to control actions based on moisture levels
120   switch (state) {
```

```
    case 0: // BELOW_THRESHOLD
      if (percentage1 < 60.0 || percentage2 < 60.0) {
        Serial.println("Moisture below 60%: Activating pump and moving axes forward");

        // Move X-axis forward
        digitalWrite(dirPinX1, LOW);
        digitalWrite(dirPinX2, LOW);
        for (int i = 0; i < numRotationsSensor1_X; i++) {
          simpleMoveX(stepPinX1, stepPinX2, 200); // 200 steps per motor for 1 full rotation
        }
        delay(2000); // Wait 2 seconds

        // Move Y-axis forward
        digitalWrite(dirPinY, LOW);
        digitalWrite(enablePinX, HIGH); // Disable X-axis motors
        digitalWrite(enablePinY, LOW); // Enable Y-axis motor
        for (int i = 0; i < numRotationsSensor1_Y; i++) {
          simpleMoveY(stepPinY, 200); // 200 steps for 1 full rotation
        }
        delay(2000); // Wait 2 seconds

        // Move Z-axis forward
        digitalWrite(dirPinZ, LOW);
        digitalWrite(enablePinY, HIGH); // Disable Y-axis motor
        digitalWrite(enablePinZ, LOW); // Enable Z-axis motor
        for (int i = 0; i < numRotationsSensor1_Z; i++) {
          simpleMoveZ(stepPinZ, 200); // 200 steps for 1 full rotation
        }

        // Move axes backward
        digitalWrite(dirPinZ, HIGH);
        digitalWrite(enablePinZ, LOW); // Enable Z-axis motor
        for (int i = 0; i < numRotationsSensor1_Z; i++) {
          simpleMoveZ(stepPinZ, 200); // 200 steps for 1 full rotation
        }

        digitalWrite(dirPinY, HIGH);
        digitalWrite(enablePinY, LOW); // Enable Y-axis motor
        for (int i = 0; i < numRotationsSensor1_Y; i++) {
          simpleMoveY(stepPinY, 200); // 200 steps for 1 full rotation
        }

        digitalWrite(dirPinX1, HIGH);
        digitalWrite(dirPinX2, HIGH);
        digitalWrite(enablePinY, HIGH); // Disable Y-axis motor
        digitalWrite(enablePinX, LOW); // Enable X-axis motors
        for (int i = 0; i < numRotationsSensor1_X; i++) {
          simpleMoveX(stepPinX1, stepPinX2, 200); // 200 steps per motor for 1 full rotation
        }

        state = 1; // Move to next state
      }
      break;

    case 1: // SENSOR_1_BACKWARD_DONE
      Serial.println("Sensor 1: Axes moved forward and backward, waiting before proceeding to
    Sensor 2");
      delay(10000); // Wait 10 seconds before moving to Sensor 2
      state = 2; // Move to next state
      break;

```

```
    case 2: // SENSOR_2_FORWARD
      if (percentage2 < 60.0) {
        Serial.println("Sensor 2: Moisture below 60%: Activating pump and moving axes forward");

        // Move X-axis forward
        digitalWrite(dirPinX1, LOW);
        digitalWrite(dirPinX2, LOW);
        for (int i = 0; i < numRotationsSensor2_X; i++) {
          simpleMoveX(stepPinX1, stepPinX2, 200); // 200 steps per motor for 1 full rotation
        }
        delay(2000); // Wait 2 seconds

        // Move Y-axis forward
        digitalWrite(dirPinY, LOW);
        digitalWrite(enablePinX, HIGH); // Disable X-axis motors
        digitalWrite(enablePinY, LOW); // Enable Y-axis motor
        for (int i = 0; i < numRotationsSensor2_Y; i++) {
          simpleMoveY(stepPinY, 200); // 200 steps for 1 full rotation
        }
        delay(2000); // Wait 2 seconds

        // Move Z-axis forward
        digitalWrite(dirPinZ, LOW);
        digitalWrite(enablePinY, HIGH); // Disable Y-axis motor
        digitalWrite(enablePinZ, LOW); // Enable Z-axis motor
        for (int i = 0; i < numRotationsSensor2_Z; i++) {
          simpleMoveZ(stepPinZ, 200); // 200 steps for 1 full rotation
        }

        // Move axes backward
        digitalWrite(dirPinZ, HIGH);
        digitalWrite(enablePinZ, LOW); // Enable Z-axis motor
        for (int i = 0; i < numRotationsSensor2_Z; i++) {
          simpleMoveZ(stepPinZ, 200); // 200 steps for 1 full rotation
        }

        digitalWrite(dirPinY, HIGH);
        digitalWrite(enablePinY, LOW); // Enable Y-axis motor
        for (int i = 0; i < numRotationsSensor2_Y; i++) {
          simpleMoveY(stepPinY, 200); // 200 steps for 1 full rotation
        }

        digitalWrite(dirPinX1, HIGH);
        digitalWrite(dirPinX2, HIGH);
        digitalWrite(enablePinY, HIGH); // Disable Y-axis motor
        digitalWrite(enablePinX, LOW); // Enable X-axis motors
        for (int i = 0; i < numRotationsSensor2_X; i++) {
          simpleMoveX(stepPinX1, stepPinX2, 200); // 200 steps per motor for 1 full rotation
        }

        state = 3; // Move to next state
      }
      break;

    case 3: // SENSOR_2_BACKWARD_DONE
      Serial.println("Sensor 2: Axes moved forward and backward, waiting before proceeding");
      delay(10000); // Wait 10 seconds before proceeding
      state = 0; // Reset state to check moisture level again
      break;
  }
```

```
241
242   temperatureFanControl();
243   lightControl();
244 }
245 if (Serial.available() > 0) {
246     char command = Serial.read();
247     float distance = 5;
248     switch (command) {
249       case 'X':
250         bool forward = TRUE;
251         moveX(stepPin1, stepPin2, distance, forward)// Code to move X motor forward
252         break;
253       case 'x':
254         bool forward = FALSE;
255         moveX(stepPin1, stepPin2, distance, forward)// Code to move X motor backward
256         break;
257       case 'Y':
258         bool forward = TRUE;
259         moveY(int stepPin, int steps)// Code to move Y motor forward
260         break;
261       case 'y':
262         bool forward = FALSE;
263         moveY(int stepPin, int steps)// Code to move Y motor backward
264         break;
265       case 'Z':
266         bool forward = TRUE;
267         moveZ(int stepPin, int steps)// Code to move Z motor forward
268         break;
269       case 'z':
270         bool forward = FALSE;
271         moveZ(int stepPin, int steps)// Code to move Z motor backward
272         break;
273       case's1':
274         int speed = 10000;
275         break
276       case's2':
277         int speed = 1000;
278         break
279       case's3':
280         int speed = 100;
281         break
282       default:
283         break;
284     }
285   }
286   }
287
288   temperatureFanControl();
289   lightControl();
290 }
291
292 // Function to move both stepper motors of the X-axis for a given distance and direction
293 void moveX(int stepPin1, int stepPin2, float distance, bool forward, int speed) {
294   // Calculate the number of steps required based on the distance
295   float stepsPerMM = 2000; // 200 steps per revolution, and 10 revolutions per mm (0.1mm per
      revolution)
296   int steps = distance * stepsPerMM;
297
298   // Set the direction of movement based on the 'forward' parameter
299   digitalWrite(dirPinX1, forward ? LOW : HIGH); // Assuming dirPinX1 is one direction pin for
      X-axis
```

```
300     digitalWrite(dirPinX2, forward ? LOW : HIGH); // Assuming dirPinX2 is the other direction
          pin for X-axis
301
302     // Loop through the calculated number of steps
303     for (int i = 0; i < steps; i++) {
304       // Trigger an impulse to move the motors
305       digitalWrite(stepPin1, HIGH);
306       digitalWrite(stepPin2, HIGH);
307       delayMicroseconds(speed); // Adjust delay as needed for motor speed
308       digitalWrite(stepPin1, LOW);
309       digitalWrite(stepPin2, LOW);
310       delayMicroseconds(speed); // Adjust delay as needed for motor speed
311     }
312   }
313
314   // Function to move the stepper motor of the Y-axis for a given distance and direction
315   void moveY(int stepPin, float distance, bool forward) {
316     // Calculate the number of steps required based on the distance
317     float stepsPerMM = 2000; // 200 steps per revolution, and 10 revolutions per mm (0.1mm per
          revolution)
318     int steps = distance * stepsPerMM;
319
320     // Set the direction of movement based on the 'forward' parameter
321     digitalWrite(dirPinY, forward ? LOW : HIGH); // Assuming dirPinY is the direction pin for Y-
          axis
322
323     // Loop through the calculated number of steps
324     for (int i = 0; i < steps; i++) {
325       // Trigger an impulse to move the motor
326       digitalWrite(stepPin, HIGH);
327       delayMicroseconds(speed); // Adjust delay as needed for motor speed
328       digitalWrite(stepPin, LOW);
329       delayMicroseconds(speed); // Adjust delay as needed for motor speed
330     }
331   }
332
333   // Function to move the stepper motor of the Z-axis for a given distance and direction
334   void moveZ(int stepPin, float distance, bool forward) {
335     // Calculate the number of steps required based on the distance
336     float stepsPerMM = 2000; // 200 steps per revolution, and 10 revolutions per mm (0.1mm per
          revolution)
337     int steps = distance * stepsPerMM;
338
339     // Set the direction of movement based on the 'forward' parameter
340     digitalWrite(dirPinZ, forward ? LOW : HIGH); // Assuming dirPinZ is the direction pin for Z-
          axis
341
342     // Loop through the calculated number of steps
343     for (int i = 0; i < steps; i++) {
344       // Trigger an impulse to move the motor
345       digitalWrite(stepPin, HIGH);
346       delayMicroseconds(speed); // Adjust delay as needed for motor speed
347       digitalWrite(stepPin, LOW);
348       delayMicroseconds(speed); // Adjust delay as needed for motor speed
349     }
350   }
351
352   // Function to move both stepper motors of the X-axis simultaneously
353   void simpleMoveX(int stepPin1, int stepPin2, int steps) {
354     for (int i = 0; i < steps; i++) {
355       digitalWrite(stepPin1, HIGH);
```

```
356       digitalWrite(stepPin2, HIGH);
357       delayMicroseconds(1000);
358       digitalWrite(stepPin1, LOW);
359       digitalWrite(stepPin2, LOW);
360       delayMicroseconds(1000);
361     }
362   }
363
364   // Function to move the stepper motor of the Y-axis
365   void simpleMoveY(int stepPin, int steps) {
366     for (int i = 0; i < steps; i++) {
367       digitalWrite(stepPin, HIGH);
368       delayMicroseconds(1000);
369       digitalWrite(stepPin, LOW);
370       delayMicroseconds(1000);
371     }
372   }
373
374   // Function to move the stepper motor of the Z-axis
375   void simpleMoveZ(int stepPin, int steps) {
376     for (int i = 0; i < steps; i++) {
377       digitalWrite(stepPin, HIGH);
378       delayMicroseconds(1000);
379       digitalWrite(stepPin, LOW);
380       delayMicroseconds(1000);
381     }
382   }
383
384   void temperatureFanControl() {
385     float temperature = sht31.readTemperature();
386
387     if (!isnan(temperature)) {
388       Serial.print("Temperature: ");
389       Serial.print(temperature);
390       Serial.println("  C ");
391
392       if (temperature > 26.7) {
393         // Turn on all the fans
394         digitalWrite(relayPinFans, HIGH);
395         Serial.println("Fans turned on");
396       } else {
397         // Turn off all the fans
398         digitalWrite(relayPinFans, LOW);
399         Serial.println("Fans turned off");
400       }
401     } else {
402       Serial.println("Failed to read from SHT31 sensor!");
403     }
404   }
405
406   void lightControl() {
407     unsigned long currentMillis = millis();
408     static bool lightsOn = false; // Variable to track the current state of the lights
409     static unsigned long lightsStartTime = 0; // Time when the lights were turned on
410
411     // If lux is above 700, turn off the lights
412     if (tsl.getLuminosity(TSL2591_VISIBLE) > 700) {
413       digitalWrite(relayPinLights, LOW); // Turn off the lights
414       lightsOn = false; // Update lights state
415       lightsStartTime = 0; // Reset lights start time
416     }
```

```
417     // Otherwise, check light on/off cycle
418     else {
419       // If lights are on and the on time has elapsed, turn them off
420       if (lightsOn && (currentMillis - lightsStartTime >= 30000)) {
421         digitalWrite(relayPinLights, LOW); // Turn off the lights
422         lightsOn = false; // Update lights state
423       }
424       // If lights are off and the off time has elapsed, turn them on
425       else if (!lightsOn && (currentMillis - lightsStartTime >= 35000)) {
426         digitalWrite(relayPinLights, HIGH); // Turn on the lights
427         lightsOn = true; // Update lights state
428         lightsStartTime = currentMillis; // Update lights start time
429       }
430     }
431   }
```

Python code for HMI for Raspberry

```python
import serial
import tkinter as tk
import time

ser = serial.Serial('/dev/ttyUSB0', 115200)

# Création de la fenêtre principale
root = tk.Tk()
root.title("Menu Principal")

# Definition de certaines variables globales
z_position = -6
y_position = 3
x_position = 4
incr = 0.1
speed = 10
X_coord = 0
Y_coord = 0
Z_coord = 0

# Création des StringVar pour stocker les coordonnées
str_x_position = tk.StringVar()
str_y_position = tk.StringVar()
str_z_position = tk.StringVar()

# Initialisation des StringVar avec les valeurs initiales des coordonnées
str_x_position.set(str(x_position))
str_y_position.set(str(y_position))
str_z_position.set(str(z_position))

# Fonction pour déplacer le chariot selon l'axe X
def move_x(direction):
    global x_position
    global speed
    if direction == "forward":
        x_position += incr*speed
        ser.write(b'X')
    elif direction == "backward":
        x_position -= incr*speed
        ser.write(b'x')
    x_position = round(x_position,1)
    str_x_position.set(str(round(x_position,1)))
    print(f"X={x_position}")


# Fonction pour déplacer le chariot selon l'axe Y
def move_y(direction):
    global y_position
    global incr
    if direction == "forward":
        y_position += incr*speed
        ser.write(b'Y')
    elif direction == "backward":
        y_position -= incr*speed
        ser.write(b'y')
    y_position = round(y_position,1)
    str_y_position.set(str(round(y_position,1)))
    print(f"Y={y_position}")

```

```
60
61  # Fonction pour déplacer le chariot selon l'axe Z
62  def move_z(direction):
63      global z_position
64      global incr
65      if direction == "up":
66          z_position += incr*speed
67          ser.write(b'Z')
68          print(f"Z={z_position}")
69      elif direction == "down":
70          z_position -= incr*speed
71          ser.write(b'z')
72          print(f"Z={z_position}")
73      z_position = round(z_position,1)
74      str_z_position.set(str(round(z_position,1)))
75      print(f"Z={z_position}")
76
77
78  def r0():
79      while(z_position != 0):
80          move_z('up')
81      print('deplacement en Z fini')
82      while(y_position != 0):
83          move_y('backward')
84      print('deplacement en Y fini')
85      while(x_position != 0):
86          move_x('backward')
87      print('deplacement en X fini')
88
89  def goTo(X_coord, Y_coord, Z_coord):
90      global x_position
91      global y_position
92      global z_position
93      while (z_position != 0):
94          move_z('up')
95      print('deplacement en Z fini')
96      time.sleep(1.5)
97      while (y_position != 0):
98          move_y('backward')
99      print('deplacement en Y fini')
100     time.sleep(1.5)
101     if X_coord > x_position:
102         while (x_position != X_coord):
103             move_x('forward')
104     else:
105         while (x_position != X_coord):
106             move_x('backward')
107     print('deplacement en X fini')
108     time.sleep(1.5)
109     if Y_coord > y_position:
110         while (y_position != Y_coord):
111             move_y('forward')
112     else:
113         while (y_position != Y_coord):
114             move_y('backward')
115     print('deplacement en Y fini')
116     time.sleep(1.5)
117     if Z_coord > z_position:
118         while (z_position != Z_coord):
119             move_y('up')
120     else:
```

```
121             while (z_position != Z_coord):
122                 move_z('down')
123         print('deplacement en Z fini')
124         time.sleep(1.5)
125 
126     def save_coords():
127         global X_coord
128         global Y_coord
129         global Z_coord
130         X_coord = float(entry_x_position.get())
131         Y_coord = float(entry_y_position.get())
132         Z_coord = float(entry_z_position.get())
133         goTo(X_coord, Y_coord, Z_coord)
134 
135 
136     def showPosition():
137         print(f'z={z_position}')
138         print(f'y={y_position}')
139         print(f'x={x_position}')
140 
141 
142     def speedRate(s):
143         global speed
144         if s == 1:
145             #ser.write(b's1')
146             speed = 1
147         elif s == 2:
148             #ser.write(b's2')
149             speed = 10
150         else:
151             #ser.write(b's3')
152             speed = 100
153 
154     # Fonction pour afficher la page d'affichage des données
155     def open_data_display_window():
156         data_window = tk.Toplevel()
157         data_window.title("Affichage des données")
158 
159         # Labels pour afficher les données
160         label_temperature = tk.Label(data_window, text="Température ambiante: ")
161         label_temperature.grid(row=0, column=0, padx=10, pady=5)
162 
163         label_humidity = tk.Label(data_window, text="Humidité du sol: ")
164         label_humidity.grid(row=1, column=0, padx=10, pady=5)
165 
166         label_pump_status = tk.Label(data_window, text="État de la pompe: ")
167         label_pump_status.grid(row=2, column=0, padx=10, pady=5)
168 
169         label_fan_status = tk.Label(data_window, text="État des ventilateurs: ")
170         label_fan_status.grid(row=3, column=0, padx=10, pady=5)
171 
172         label_light_status = tk.Label(data_window, text="État des lumières: ")
173         label_light_status.grid(row=4, column=0, padx=10, pady=5)
174 
175         # Emplacements réservés pour afficher les valeurs des données
176         entry_temperature = tk.Entry(data_window)
177         entry_temperature.grid(row=0, column=1, padx=10, pady=5)
178 
179         entry_humidity = tk.Entry(data_window)
180         entry_humidity.grid(row=1, column=1, padx=10, pady=5)
181 
```

```
    entry_pump_status = tk.Entry(data_window)
    entry_pump_status.grid(row=2, column=1, padx=10, pady=5)

    entry_fan_status = tk.Entry(data_window)
    entry_fan_status.grid(row=3, column=1, padx=10, pady=5)

    entry_light_status = tk.Entry(data_window)
    entry_light_status.grid(row=4, column=1, padx=10, pady=5)


# Fonction pour afficher la page de contrôle des moteurs
def open_motor_control_window():
    global entry_x_position
    global entry_y_position
    global entry_z_position

    motor_window = tk.Toplevel()
    motor_window.title("Contrôle des moteurs")

    # Labels pour afficher la position du chariot
    label_x_position = tk.Label(motor_window, text="Trolley position X =")
    label_x_position.grid(row=0, column=0, padx=10, pady=5)

    label_y_position = tk.Label(motor_window, text="Trolley position Y =")
    label_y_position.grid(row=1, column=0, padx=10, pady=5)

    label_z_position = tk.Label(motor_window, text="Trolley position Z =")
    label_z_position.grid(row=2, column=0, padx=10, pady=5)

    # Emplacements réservés pour demander les valeurs des positions
    entry_x_position = tk.Entry(motor_window)
    entry_x_position.grid(row=0, column=1, padx=10, pady=5)

    entry_y_position = tk.Entry(motor_window)
    entry_y_position.grid(row=1, column=1, padx=10, pady=5)

    entry_z_position = tk.Entry(motor_window)
    entry_z_position.grid(row=2, column=1, padx=10, pady=5)

    # Création des boutons pour contrôler les moteurs
    btn_x_forward = tk.Button(motor_window, text="X+", command=lambda: move_x('forward'))
    btn_x_forward.grid(row=3, column=0, padx=10, pady=5)

    btn_x_backward = tk.Button(motor_window, text="X-", command=lambda: move_x('backward'))
    btn_x_backward.grid(row=3, column=1, padx=10, pady=5)

    btn_y_forward = tk.Button(motor_window, text="Y+", command=lambda: move_y('forward'))
    btn_y_forward.grid(row=4, column=0, padx=10, pady=5)

    btn_y_backward = tk.Button(motor_window, text="Y-", command=lambda: move_y('backward'))
    btn_y_backward.grid(row=4, column=1, padx=10, pady=5)

    btn_z_up = tk.Button(motor_window, text="Z+", command=lambda: move_z('up'))
    btn_z_up.grid(row=5, column=0, padx=10, pady=5)

    btn_z_down = tk.Button(motor_window, text="Z-", command=lambda: move_x('down'))
    btn_z_down.grid(row=5, column=1, padx=10, pady=5)

    btn_raz = tk.Button(motor_window, text="X0 Y0 Z0", command=lambda: r0())
    btn_raz.grid(row=6, column=0, padx=10, pady=5)

```

```
    btn_pos = tk.Button(motor_window, text="Xs Ys Zs", command=lambda: showPosition())
    btn_pos.grid(row=6, column=1, padx=10, pady=5)

    # Affichage de la position en X
    aff_x_position = tk.Label(motor_window, text='X = ')
    aff_x_position.grid(row=7, column=0, padx=5, pady=5)
    label_x_position_val = tk.Label(motor_window, textvariable=str_x_position)
    label_x_position_val.grid(row=7, column=1, padx=10, pady=5)

    # Affichage de la position en Y
    aff_y_position = tk.Label(motor_window, text='Y = ')
    aff_y_position.grid(row=8, column=0, padx=5, pady=5)
    label_y_position_val = tk.Label(motor_window, textvariable=str_y_position)
    label_y_position_val.grid(row=8, column=1, padx=10, pady=5)

    # Affichage de la position en Z
    aff_z_position = tk.Label(motor_window, text='Z = ')
    aff_z_position.grid(row=9, column=0, padx=5, pady=5)
    label_z_position_val = tk.Label(motor_window, textvariable=str_z_position)
    label_z_position_val.grid(row=9, column=1, padx=10, pady=5)

    # Bouton pour enregistrer les coordonnées demandées
    btn_save_coord_asked = tk.Button(motor_window, text="Go to", command=save_coords)
    btn_save_coord_asked.grid(row=1, column=2, columnspan=2, padx=10, pady=5)

    # Définition de la vitesse Vitesse 1
    speed1 = tk.Button(motor_window, text="Step = 0.1mm", command=lambda: speedRate(1))
    speed1.grid(row=10, column=0, columnspan=2, padx=10, pady=5)

    # Définition de la vitesse Vitesse 2
    speed2 = tk.Button(motor_window, text="Step = 1mm", command=lambda: speedRate(2))
    speed2.grid(row=10, column=1, columnspan=2, padx=10, pady=5)

    # Définition de la vitesse Vitesse 3
    speed3 = tk.Button(motor_window, text="Step = 10mm", command=lambda: speedRate(3))
    speed3.grid(row=10, column=2, columnspan=2, padx=10, pady=5)


# Configuration du fond d'écran du menu principal
background_image = tk.PhotoImage(file="menu_background.gif")
background_label = tk.Label(root, image=background_image)
background_label.place(relwidth=1, relheight=1)

# Création des boutons pour l'affichage des données et le contrôle des moteurs sur le menu
    principal
btn_data_display = tk.Button(root, text="Affichage des données", command=
    open_data_display_window)
btn_data_display.place(relx=0.3, rely=0.4, relwidth=0.2, relheight=0.2)

btn_motor_control = tk.Button(root, text="Contrôle des moteurs", command=
    open_motor_control_window)
btn_motor_control.place(relx=0.5, rely=0.4, relwidth=0.2, relheight=0.2)

# Boucle principale de l'interface graphique
root.mainloop()
```