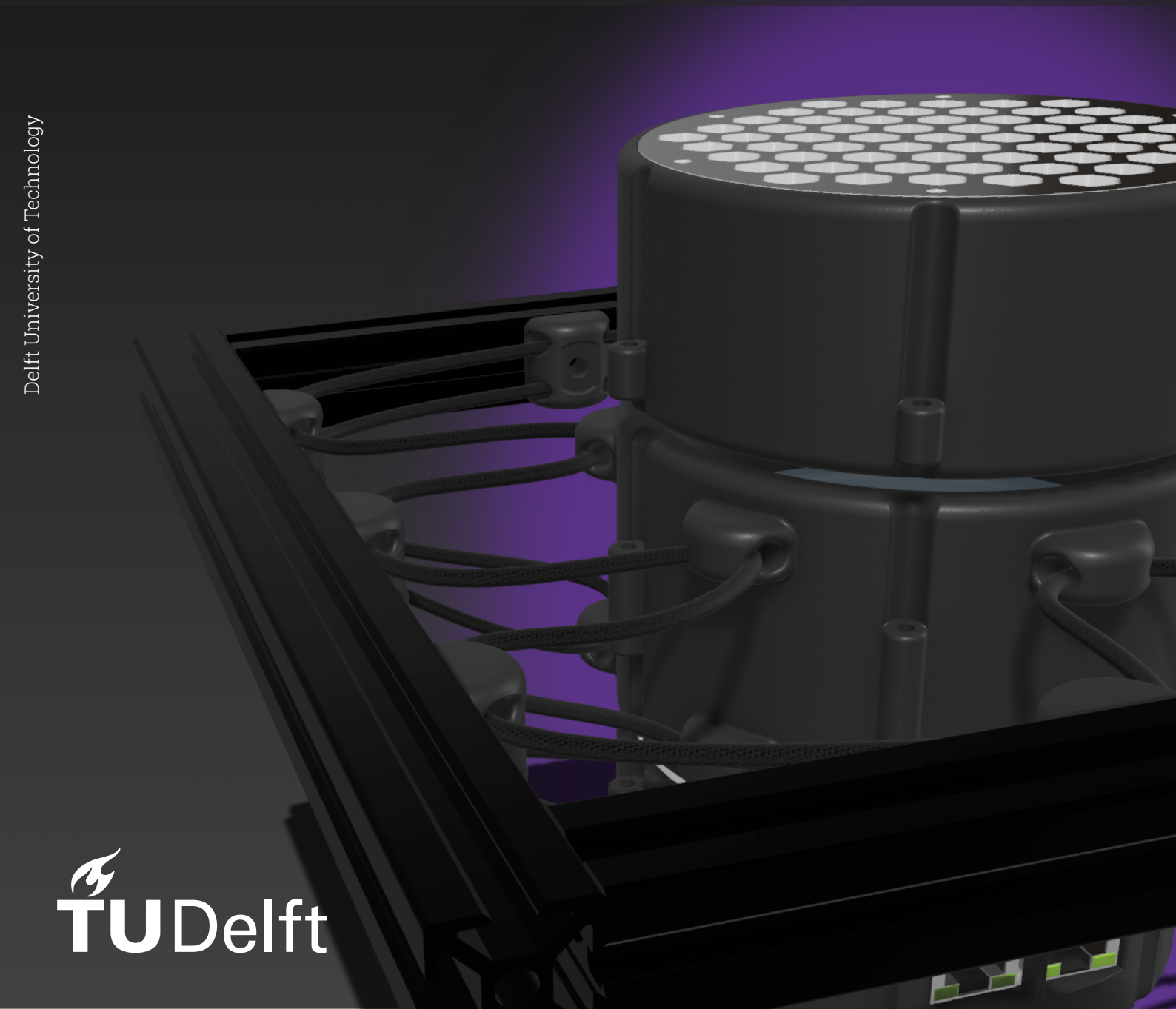
UVC Seed Sterilization BSc Thesis

Software and Control

EE3L11: Bachelor Graduation Project

Erman Ergül
Erik van Weelderen



Delft University of Technology

TU Delft

UVC Seed Sterilization BSc Thesis

Software and Control

by

Erman Ergül
Erik van Weelderen

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 27, 2023 at 15:30.

Student number:	5334640 (E. Ergül)	
	5315115 (E.H. van Weelderen)	
Project duration:	24 April, 2023 - 30 June, 2023	
Thesis committee:	Dr. M. Babaie,	TU Delft, jury chair
	Dr. ing. H. van Zeijl,	TU Delft, supervisor
	Dr. S. Izadkhast,	TU Delft
	Prof. dr. ir. J. van Turnhout	TU Delft
	Drs. L. Wymenga,	TU Delft, supervising PhD candidate
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft	

with contribution from the entire UVC Seed Sterilization group
Bsc. Electrical Engineering 2023 BAP group M:

E. Ergül	5334640
R.W.L Imbens	5155940
L.C. Klootwijk	5155940
M. Mazurovs	5050545
D.O. Schat	5169801
E.H. van Weelderen	5315115

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

In this thesis for the Bachelor's Degree in Electrical Engineering, a Control Unit (PCB and software) is designed for inside the UVC Seed Disinfection Machine of Team UVO. The machine consists of four modules: the Control Unit, the LED Driver, the motor controller and the power supply. The Control Unit allows a user to input the intensity per wavelength (for possible wavelengths: 255 nm, 275 nm, 285 nm and 395 nm), the exposure time and the motor speed.

The design of the machine, including all the modules, is aimed at achieving the optimal wavelength for inactivation and maximum and uniform irradiance, with the ability of changing radiation settings according to the desire of the user.

The Control Unit manages communication with the other modules, data storage, the User Interface, safety checks and system enabling. The thesis covers the design choices regarding the entire design, with an in-depth analysis of the hardware implemented safety checks, the graphical user interface and the design of the communication protocol.

Due to difficulties regarding uploading the code onto the PCB, not every developed functionality could be tested or implemented. However, the functionalities that were tested, did perform as expected. In addition to this, after the thesis has been submitted, more time will be spent on debugging the PCB, implementing and testing its features.

The objective for the graduation project of Team UVO is to provide a proof-of-concept of disinfecting cabbage seeds (*Brassica oleracea capitata*) from Alternaria using UVC LEDs. This thesis describes design process of the Control Unit module. The results of the decontamination process are provided in Appendix F.

Preface

For this Bachelor Graduation Project of Electrical Engineering, we, Team UVO, have worked on a device to disinfect seeds. It was quite a challenge, and we knew from the start what we were getting ourselves into. However, this challenge was what drove us, pushed us to go further and to reach for the best possible. And so we did.

But not without help.

First and foremost, we would like to extend our gratitude to dr. ing. H. van Zeijl, drs. L. Wymenga and Prof. dr. ir. J. van Turnhout. Without you, this project would not have existed. Your guidance and enthusiasm for the project has motivated us from the first meeting.

Additionaly, we would like to thank M. Schumacher and A.M.J. Slats for allowing us to make use of their working equipment and supplies and allowing us to work inside the Tellegen Hall.

Finally, we would like to thank Rik Imbens, Lucas Klootwijk, Maxim Mazurovs and Daan Schat for joining us in this amazing project.

It took us long hard-working days, every possible drink at the coffee machine, and lots of fun, so hereby we proudly present our Bachelor's Graduation Thesis: Software and Control.

Erman Ergül
Erik van Weelderen
Delft, July 2023

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
CAD	Computer Aided Design
CS	Chip Select, used to set a module as target in SPI
DNA	Deoxyribonucleic acid
DNP	Do-not-place
EMR	Electromagnetic Radiation
GPIO	General Purpose Input Output
GUI	Graphical User Interface
IC	Integrated-Circuit
I^2C	Inter-Integrated Circuit
LED	Light-Emitting Diode
LED_OC_FLAG	LED Driver Overcurrent Flag
LED_OO_FLAG	LED Driver Overozone Flag
LED_OT_FLAG	LED Driver Overtemperature Flag
MC_OC_FLAG	Motor Controller Overcurrent Flag
PCB	Printed Circuit Board
PoC	Proof-of-Concept
PoR	Programme of Requirements
PS_OC_FLAG	Power Supply Unit Overcurrent Flag
PSD	Power spectral density
RNA	Ribonucleic acid
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
UVGI	Ultraviolet germicidal irradiation
UVR	Ultraviolet Radiation
WPE	Wall-plug efficiency
UV	Ultraviolet

Glossary

Term	Definition
Base Protocol	The protocol on which the communication protocol is built on top of (i.e. UART, I^2C , SPI)
Controller	The module taking the lead in communication
Communication Protocol	The protocol the systems uses defining how to communicate
Target	The module that follows in the communication protocol

Symbols

Symbol	Definition	Unit
T	Transmission	[-]

Symbol	Definition	Unit
I	Irradiance of light exiting the solution	[W/m^2]
I_0	Irradiance of light entering the solution	[W/m^2]
A	Absorbance	[-]
l	Thickness of the solution	[cm]
c	Concetration of solute	[moles/liter]
P_{opt}	Optical power emitted from source	[W]
P_{in}	Total power dissipated by the source	[W]
WPE	Wall-plug efficiency	[-]
L	Total thickness of the material	[mm]
V_{gs}	Gate-to-source voltage	[V]
V_{Th}	Threshold voltage of a MOSFET	[V]
t	Time	[s]
ε	molar absorptivity	[liters/mole-cm]
η_{eff}^{Hg}	optical power efficiency mercury lamp	[-]
α	Absorption coefficient per unit length	[mm^{-1}]
θ_{inner}	Angle of inner ring of the LEDs on the LED panel	[rad]
θ_{outer}	Angle of outer ring of the LEDs on the LED panel	[rad]

1 Introduction

Seeds are the basis of the agricultural industry and a source of a great portion of humanities food supply. With the plants, fruits and vegetables that grow out of these seeds, humans can feed themselves. Because of this very reason, the seed industry has specialized itself in making the perfect seeds for every species. Rijk Zwaan is one of the largest vegetable distribution companies in the Netherlands [1]. Their main task is also to provide the healthiest and best seeds, as demanded by the market, whereby the market demand varies per region and country.

One of their foremost challenges is to disinfect every single one of the seeds that is exported from all sorts of pathogens, such as fungi, bacteria, and viruses. If the process of disinfection is not done properly, the plants will grow contaminated by these pathogens and fall sick. This can have disastrous consequences for the harvest. It may eventually also have an impact on how much people have to pay for their vegetables. The reason for this is that disinfecting seeds is one of the first steps in the whole chain of growing plants and selling them.

In a presentation [2], given by Rijk Zwaan at their headquarters in de Lier, Netherlands, it was explained that the current method of disinfecting seeds is to dip the seeds in a hot water bath. In this way, the pathogens are being activated and eventually killed because of the heat of the water. However, this method has two serious drawbacks. Firstly, drying the seeds after the process requires a lot of energy. Hence, it decreases the total efficiency and commercial feasibility. Secondly, the seeds can undergo partial or complete degradation due to the thermal stress. This has a significant impact on the downstream supply chain. Thirdly, this process is time-consuming, which makes it undesirable.

Due to complications that arise with the conventional hot water treatment, enterprises such as Rijk Zwaan have started to invest in alternative disinfection techniques. The use of Ultraviolet (UV) light is one of such techniques [3].

1.1. Project Objective

Although, the traditional UV light tubes are potential candidates for this application. However, these contain the toxic element mercury. That is why, with the rise of UV light-emitting diodes (LEDs), research has been launched in the form of an Electrical Engineering Graduation Project. The main objective is to provide a proof-of-concept (PoC) of disinfecting seeds using UV LEDs. It was opted to make the device as a product that could be further developed into a device that may be used commercially. During the duration of this project, it remained in an experimental phase. It was decided with Rijk Zwaan and the supervisors that the species that would be disinfected is cabbage seed (*Brassica oleracea capitata*) and that the pathogen species that would be focussed on is Alternaria, a fungus. This is a common fungus, but also one of the hardest fungi to inactivate [2]. Being able to disinfect the cabbage

seeds from this fungus increases the probability of disinfecting other pathogens.

To properly distribute the tasks, the group of six students split up into three subgroups. The LED Driving and Sensing group [4] is responsible for designing the system that provides the UV light to the seeds and ensures that the relevant parameters are being monitored with the use of sensors. The Mechanics subgroup [5] provides the whole system with power and designs the mechanical side of the system. Finally, the last subgroup is covered in the thesis: the design of the Control Unit.

1.2. State-of-the-Art Analysis

To properly approach this research, it must be explained first what UV light is. Also, its disinfecting properties must also be explained.

1.2.1. Ultraviolet light in the light spectrum

UV light originates from the electromagnetic radiation (EMR) that is emitted by the sun. This radiation contains several wavelengths, ranging from shorter wavelengths that are high in energy to long-wavelength that are low in energy. Ultraviolet radiation (UVR) waves range from 200 nm to 400 nm. UVR can be further divided into four different classes. Firstly, there is UVA, ranging from 320 nm to 400 nm. Secondly, UVB ranging from 290 nm to 320 nm. Thirdly, the class with the most energy, UVC ranging from 200 nm to 290 nm. This class however does not penetrate into the atmosphere due to the ozone layer. Finally, the last class is UV Vacuum, which ranges from 100 nm to 200 nm [6]–[8]. The EMR and UVR spectrum are illustrated in detail in Figure 1.1.

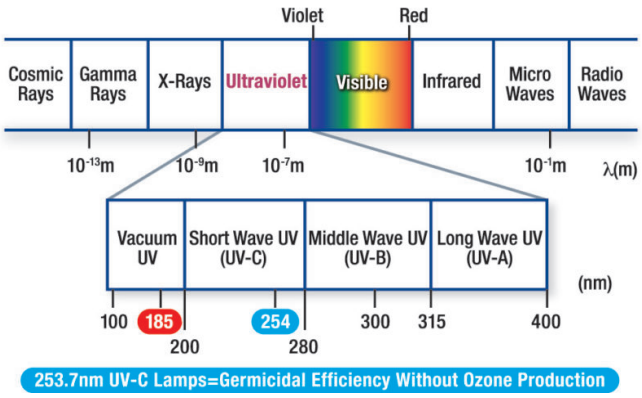


Figure 1.1: This figure illustrates the electromagnetic radiation spectrum with the UVR in detail. [8]

1.2.2. Germicidal properties of Ultraviolet light

Pathogens have not developed resistance against UVC waves, as the ozone layer provides protection against UVR [9]. This resulted in a phenomenon called ultraviolet germicidal irradiation (UVGI), which is defined as the germicidal effectiveness of UV. UVC is dangerous for the pathogens, as it damages proteins, ribonucleic acid (RNA), and deoxyribonucleic acid (DNA) [10]. Furthermore, it has been proven before that UVGI is an effective method for inactivating Alternaria on for example tomatoes [11]. This indicates that the proposed method of using UVC LEDs has a high chance of succeeding.

As Alternaria is a fungus, the most sensitive target is its DNA. DNA has adenine, cytosine, guanine, and thymine as its bases [12]. UVC radiation can inactivate pathogens by creating cross-links between nucleic acids. These cross-links are called intrastrand cyclobutyl-pyrimidine dimers and are the causes of cell death or mutations. The first dimers formed are the thymine dimers and secondly the cytosine dimers. Furthermore, UVR can also cause photohydration reactions, where cytosine and uracil can bond with elements of water molecules. This phenomenon is independent of the UV wavelength. Figure 1.2 illustrates the formation of thymine dimers between two thymine nucleotides that are adjacent to each other, this is called a thymine doublet. Thymine dimers have a stronger cohesion than hydrogen bonds that normally exist in DNA helices [10].

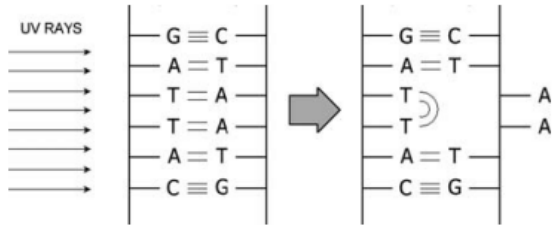


Figure 1.2: Thymine dimers are created by the absorption of UVC radiations in adjacent thymine nucleotides [10].

UV Absorption Spectrum of proteins, DNA and RNA

The absorption spectrum describes the absorptivity of certain molecules over the EMR spectrum. Whenever UV is absorbed by a molecule, it enters an excited state. The probability of a molecule entering the excited state is strictly related to the intensity of the absorption band and the probability of a photon of the right energy being present. The spectra of molecules can be found by beaming light through a solution containing the molecules of interest and comparing it to a clean solution (no molecules). The transmittance T follows from this measurement and is defined as [10]:

$$T = \frac{I}{I_0} \tag{1.1}$$

where

I = irradiance of light exiting the solution, W/m^2
I_0 = irradiance of light entering the solution, W/m^2

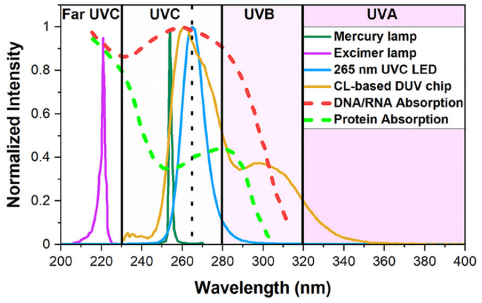
From this, Beer's law constitutes a relationship between the transmittance (Equation (1.1)) and the absorbance, A [10]:

$$T = \frac{I}{I_0} = 10^{-A} \tag{1.2}$$

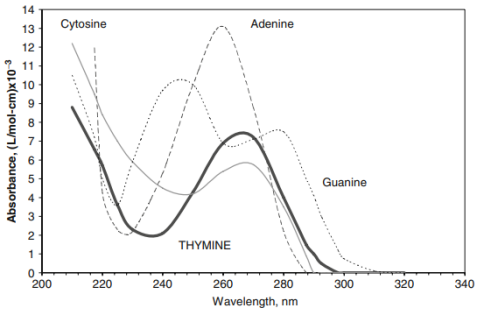
Absorbance is defined as $A = \varepsilon lc$, where

ε = molar absorptivity, liters/mole-cm
l = thickness of the solution, cm
c = concentration of solute, moles/liter

For the absorbance spectra of the four nucleotides and a normalized absorbance spectra of proteins, DNA and RNA refer to Figure 1.3. It can be seen that the peak of the absorbance spectrum of DNA is around 260 nm, with the absorbance of thymine peaking at 265 nm.



(a) Absorbance spectrum of proteins, DNA and RNA [9]. Note that the y-axis is given in normalized intensity (normalized absorbance)



(b) Absorbance spectrum of the four nucleotides [10].

Figure 1.3: A side by side illustration of the absorption spectra of the four main nucleotides and proteins, DNA and RNA.

1.2.3. Comparison of mercury UV lamps and UV LEDs

Around 85% of the optical output of low-pressure mercury lamps is near 254 nm, which is close to the peak mentioned in Section 1.2.2 and they have an optical power efficiency of 60% [9], defined as:

$$\eta_{eff}^{Hg} = \frac{P_{opt}}{P_{in}} \tag{1.3}$$

where

P_{opt} = optical power emitted from the source, W
P_{in} = total power dissipated by the source, W

For UVC LEDs, a key parameter in defining the efficiency is the wall-plug efficiency (WPE). It is defined as the optical output power (P_{opt}) divided by the electrical input power (VI):

$$WPE = \frac{P_{opt}}{VI} \tag{1.4}$$

The WPE of LEDs oriented around the peak absorbance wavelength is around 1-4%, which is much lower than the mercury lamps. However, a lot of research is happening in the field of chip-scale UVC technologies, so it is expected that it will rise to 10% in 2025 [9].

Although mercury lamps exhibit a higher efficiency, due to its toxic properties to humans and environmental danger there is a decrease in their production and uses [9]. Furthermore, the mercury lamps need to warm up before being effective in the disinfection process, which is a waste of energy, thus not sustainable [2], [9]. The size of mercury lamps is also quite big, making them hard to package in tight devices or systems. Hereby increasing the distance from optical output power source to receiving sample [2].

1.2.4. Safety considerations of UV on humans

Despite its germicidal capabilities and other benefits, UVR can be harmful to humans when they are exposed to it too much [13]. One of the effect it can have is sunburn, but it can also lead to skin cancer eventually. Furthermore, human eyes also have some transmission for UVR, which can cause complications to the eyes such as photokeratitis, erythema of the eyelid, cataracts, solar retinopathy, and retinal damage [14]. Furthermore, another risk that comes with UV disinfection, is the generation of ozone in the process [13]. Ozone can cause damage to the respiratory system of the human body. The allowed maximum ozone concentration is 0.1 ppm $\left[200\,\mathrm{g/m^3}\right]$ [15].

Dangers to the skin

UV can penetrate the epidermis and damage keratinocytes and melanocytes, with both of them causing different kinds of skin cancer. Melanocytes is the cause of malignant melanoma of the skin, which is the most harmful type of skin cancer. UVB can cause sunburn by creating a cascade of cytokines, vasoactive and neuroactive mediators [16].

Dangers to the eye

UV light can induce a cataract, which is an opacity of the lens. Cataracts are created when the tissue that makes up the lens of the eye gets damaged. Especially UVB is associated with an increased risk in inducing a cataract. This can cause the vision of a person to become hazy or even develop blind spots [14]. Furthermore, exposure to UV can also cause retina degeneration, which can lead to the death of retinal photoreceptors such as rods and cones. This loss of photoreceptors and the loss of vision impacts the daily life of subjects that experience retina degeneration as they can lose the ability of not recognizing faces, read and find objects [14], [17].

Preventive measures

Literature ([14], [18]) states that there are measures that can be taken into account to prevent UV from damaging the skin and eyes. Clothes with a lower transmission of UV can prevent UV from penetrating through it and eventually stop UV from penetrating the epidermis. Furthermore, if a person is working hands-on with UV light sources, hand gloves can prevent from UV getting in direct contact with the hands. Furthermore, the use of AS1067 rated sunglasses can provide adequate protection to the eyes. The UV window of tinted sunglasses is very important because of pupillary dilation, thus increasing the exposure of the retina to UV. Plastic glasses can provide enough protection to protect from UVB radiation.

1.3. Thesis Layout

This thesis will guide the reader through the research done to properly select key parameters for disinfecting the cabbage seed, the process of designing hardware and software, related to the controlling of the system and monitoring the status of the system. Chapter 2 provides the programme of requirements (PoR) on which the design will be based. Chapter 3 covers the optimization of the radiation pattern. Chapter 4 explains how the Control Unit ensures that the system is safe to be switched on. In Chapter 5, the design of the user interface is described, both concerning software and hardware. Chapter 6 details how said data is conveyed to the other modules using the custom-built communication protocol. Chapter 7 explains the overall design and implementation of the Control Unit prototype. The prototype is tested, and the results are analyzed in Chapter 8. Lastly, the project is concluded in Chapter 9, followed by recommendations and future work.

2

Programme of Requirements

The programme of requirements (PoR) acts as the main leading factor in the design and development of the device. There are certain functionalities that the device must have and some functionalities that are there to make the final product more appealing to the end users. These can be distinguished as *mandatory requirements (MR)* and *trade-off requirements (ToR)*, respectively. The PoR also acts as a method of assessing the performance of the project. It must be noted that the PoR has been constructed for an experimental environment and must be reviewed when upgrading to a commercial environment, as described in Section 1.1.

Requirements that fall under the category mandatory requirements are the constraints of the design. Whether the design is a success or not is based on these requirements primarily. Therefore, one should always aim to satisfy each entry of the MR.

Trade-off requirements are requirements that the system or submodule do not necessarily have to satisfy to assess the performance of the project. However, some of these requirements might make it more appealing. The requirements shall experience a trade-off, mainly consisting of time versus project progress gain, thus how much time will it take to satisfy these requirements and how does this result in coming closer to successfully finish the project [19].

Furthermore, the requirements are divided into two different system levels. First, the PoR for the device as a whole is presented and these are general requirements which define what the device has to do. Secondly, a PoR for the Control Unit module is shown, wherein the requirements for the submodule that is described in this thesis will be shown.

2.1. System Requirements

2.1.1. Mandatory Requirements

[**SM.1**] The system must make use of UVC LEDs.

[**SM.2**] The system must inactivate Alternaria on cabbage seed.

[**SM.3**] The system must irradiate all seeds via a uniform radiation pattern

- [**SM.3.1**] The seeds must be irradiated on all sides
- [**SM.3.2**] The irradiation density must be uniform on all locations

[**SM.4**] All modules of the system must be powered from the same voltage source.

[**SM.5**] All modules of the system must be able to communicate with the Control Unit.

[**SM.6**] The system must have appropriate error detection.

[**SM.7**] The system must only be enabled when there is no error detected.

- [**SM.7.1**] The system must turn off when the UVC LEDs go beyond their optimum operating temperature.

[**SM.7.2**] The system must turn off when the ambient temperature of the seeds puts the germination chance of the seeds at risk.
[**SM.7.3**] The system must turn off when the ozone concentration in the radiation enclosure exceeds the allowed maximum concentration of 0.1 ppm $\left[200\,\mathrm{g/m^3}\right]$) [15].
[**SM.7.4**] The system must turn off when the total current draw exceeds that of the lowest maximum current rating of any component that experiences that exact amount of current flow.
[**SM.7.5**] The system must turn off when one of the modules experiences a current draw exceeding that of the lowest maximum current rating of any component in that path.

[**SM.8**] The system must be externally controllable.

[**SM.8.1**] The radiation parameters must be adjustable.

[**SM.8.1.1**] The intensity of the present wavelengths in the radiation pattern must be adjustable.
[**SM.8.1.2**] The duration of radiation must be adjustable.
[**SM.8.1.3**] The intensity of the radiation must be adjustable.

[**SM.8.2**] The system must be able to be turned off with a single switch.

[**SM.9**] The state of the system must be monitored at all times.

[**SM.9.1**] The temperature of the UVC LEDs must be monitored.
[**SM.9.2**] The temperature of the seeds must be monitored.
[**SM.9.3**] The ozone levels in the radiation enclosure must be monitored.
[**SM.9.4**] The current draw of the system must be monitored.
[**SM.9.5**] The current draw of each module must be monitored.

2.1.2. Trade-off Requirements

[**ST.1**] The system will be in a closed casing to prevent UV leakage.
[**ST.2**] The system will have an easy-access mechanism for the adding/removing the seeds.

[**ST.2.1**] The system will have an automated seed transportation mechanism.

[**ST.3**] The system will be able to inactivate more pathogens than Alternaria.
[**ST.4**] The system will be able to disinfect more seeds than cabbage seed.
[**ST.5**] Independent error detection will be implemented for finding which module triggered a system error.
[**ST.6**] The system can only turn on when the enclosure is completely sealed or closed.

2.2. Control Unit Requirements

Below is a list of the mandatory module requirements (MM) for the Control Unit:

2.2.1. Mandatory Requirements

[**MM.1**] The Control Unit must control the behaviour of the other modules in the system.
[**MM.2**] The Control Unit must communicate with all other modules of the system.
[**MM.3**] The Control Unit must check for errors.
[**MM.4**] The Control Unit must only enable the system when no errors are present.

[**MM.4.1**] The Control Unit must only be able to reset the error states with human interaction.
[**MM.4.2**] The error states must be independent signals.

[**MM.5**] The Control Unit must have a User Interface.

[**MM.5.1**] The User Interface must be able to display real-time data.
[**MM.5.2**] The User Interface must be designed as such that radiation parameters can be adjusted.
[**MM.5.3**] The Control Unit must forward the user-specified parameters to the other modules.
[**MM.5.4**] The Control Unit must display the state of the system.

[**MM.6**] The Control Unit must have data logging capabilities.

[**MM.6.1**] The user must have easy access to the data.

[**MM.7**] The Control Unit must be programmable via USB.

[**MM.8**] The Control Unit must be designed as such that debugging the system is easy.

[**MM.9**] The Control Unit must be able to reset other modules.

2.2.2. Trade-off Requirements

[**MT.1**] The Control Unit can preferably save presets of radiation parameters for sterilization of different pathogens or disinfecting different seeds.

[**MT.2**] The User Interface can be easily accessible when integrated with the entire system.

[**MT.3**] The data storage can be easily accessible when integrated with the entire system.

[**MT.4**] The User Interface can be equipped with a touchscreen display.

[**MT.5**] The Control Unit can be accessible via internet.

[**MT.5.1**] Data can be accessed via internet.

[**MT.5.2**] Radiation parameters can be adjusted via internet.

[**MT.6**] The Control Unit is preferably designed to be modular.

3

Optimization of the Radiation Pattern

3.1. Introduction

Figure 1.3 illustrates that the absorbance of different nucleotides changes per wavelength, and that protein has a much different absorption spectrum than DNA/RNA. In Section 1.2.2, it was explained how UVC radiation causes the creation of intrastrand cyclobutyl-pyrimidine dimers. These mostly exist in the form of thymine. However, when a high enough dosage is applied, other nucleotides can form these dimers [10]. Furthermore, due to the fact that lower wavelengths penetrate less deep into substances than higher wavelengths [20], a broad spectrum of UVC radiation might be beneficial to effectively inactivate the Alternaria on the cabbage seed. Furthermore, the seeds must be stationed on top of some surface. The transmission of the material of this surface is a parameter that is useful for optimal radiation of the seeds. Material choice of the environment can be taken into consideration to reflect as much of the UVR on the seeds and not lose optical energy to the materials. Altering the pattern in which the LEDs are placed, could result in a more uniform irradiation. Furthermore, applying some kind of force to the plate of the seeds might cause the seeds to move around and enable uniform irradiance.

This chapter provides the analysis of the parameters mentioned above and demonstrates simulations and graphs illustrating properties of materials and technologies. The following list provides all parameters of interest in this chapter:

- Wavelength of the UVC LEDs
- Transmission of the surface where the seeds are placed
- Reflection of UVR
- Placement of the LEDs for uniform irradiance
- Moving of the seeds on the plate for uniform irradiance

3.2. Wavelength selection of the UVC LEDs

Due to the different absorbances of the molecules present in pathogens, such as Alternaria [10], it should be analyzed which wavelength would be the most effective for inactivating them. The selection of wavelengths is important because that is where the microbes actually absorb the most UVC. Radiating around the peak wavelength results in the creation of more dimers and thus inactivating more pathogens. From the literature it was found that the absorption spectra peak at around 260 nm [8]–[10], [13], [21].

Although lots of literature has been found stating that the peak is around 260 nm to 265 nm [8]–[10], [13], [21], others present some other wavelengths. These inactivate for example viruses with 280 nm UVC. Table 3.1 shows this.

Literature	Application	Wavelength
[22]	Indoor Environment Sanitization	265nm
[23]	Sars-Cov-2	265nm or 275-280nm
[24]	Apiaceae spices	254nm
[25]	Spoilage fungi	254nm
[26]	Development of a method for testing UVC LEDs	UVA-UVB-UVC
[27]	Different species of viruses	260nm or 280nm

Table 3.1: A table illustrating which wavelengths were used by different references.

Noteworthy, is that a combination of 260 nm and 280 nm UVC LEDs was used in the research of virus disinfection [27], however it did not appear to have the same efficacy as the individual LEDs. A difference from their research and this thesis is that this thesis is intended to inactivate Alternaria, a fungus, and not a virus. Thus, the outcome of a combined LED unit might be different. Furthermore, there is little to be found about the radiation pattern and irradiance in that research. With a design that makes it possible to control all wavelengths individually, one could always have the option to only use one wavelength.

Further research [27] also demonstrates that a combination of UVA and UVC might improve the effectiveness of the disinfection process. It is stated that applying UVA to a solution for a longer period of time and then adding UVC to the equation increases the reduction of microbes.

The LED panel was designed in such a way that a combination of LEDs will be implemented. These are LEDs of the following wavelengths: 255 nm, 275 nm and a LED with a spectrum containing peaks at 285 nm and 395 nm. The latter is chosen such that it can be researched if the UVA (395 nm) can activate the fungus and the UVC (285 nm) can inactivate the fungus spores more easily. The implementation of these three LEDs means that it is possible to control every wavelength independently. Then, the optimal power spectral density (PSD) can experimentally be found for Alternaria. In addition to this, it is possible to add the PSDs of the individual LEDs to gain a total PSD with peaks in different wavelengths as well. Thus covering a large part of the total UVC spectrum [4].

3.3. Analysis of the transmission of the plating

The seeds have to be placed on a plate. As per [**SM.3.1**], the seeds must be irradiated on all sides. This implies that a lot could be gained by selecting a material for the plate that has a high transmission for UVC. By doing this, the UVC can penetrate from underneath the seeds. The absorbance of certain materials is defined by the Beer-Lambert Law of exponential decay [28]:

$$I = I_0 e^{-\alpha L} \tag{3.1}$$

where

I = irradiance of light passing through the material, W/mm^2
I_0 = irradiance of light entering the material, W/mm^2
α = absorption coefficient per unit length, mm^{-1}
L = total thickness of the material, mm

From Equation (3.1), it can be derived that thicker materials cause more attenuation of light. This means that the trade-off for choosing the thickness of the plate is between the decrease of transmission and the strength of the plate. From Figure 3.1, it can be concluded that quartz has high transmission for UVC. The other options presented in [28] are either too expensive or cause complications in the supply chain. For these reasons, the plate will be made out of quartz.

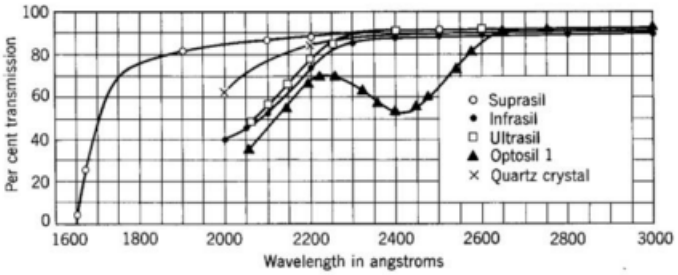


Figure 3.1: The transmission spectrum for different grades of Fused Silica: Suprasil, Infrasil, Ultrasil, Optosil 1 and Quartz crystal [28]. Note that the x-axis is given in angstroms (1000 angstroms = 100 nm)

The transmission of quartz for 260 nm is 80% [29] with a quartz plate of 1 mm. Using Equation (3.1) and Equation (1.2) and solving for α yields:

$$\alpha = \frac{-\ln(T)}{L} = \frac{-\ln(0.8)}{1} \approx 0.223 \qquad [\text{mm}^{-1}] \quad (3.2)$$

The MATLAB script in Appendix D.1 uses Equations (3.1) and (3.2) to find the transmission of quartz for 260 nm over a range of plate thicknesses. The results are presented in Figure 3.2.

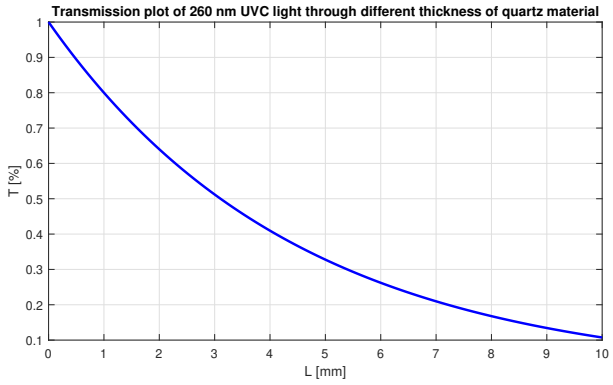


Figure 3.2: The transmission for quartz of 260 nm given over a range of plate thickness.

From Figure 3.2, it can be concluded that a quartz plate with a thickness of 2 mm has a transmission of around 65%. Due to concerns with using a thinner quartz plate being more fragile, it was opted to use a 2 mm thick quartz plate. This way, it is possible to expose the seeds from the top and bottom to UVC radiation.

3.4. Adding reflectors for optimal radiation

The total irradiation could increase by adding UVC reflecting materials in the enclosure, where the seeds will be irradiated. Figure 3.3 is the reflectance spectrum of different materials. There it can be seen that aluminum (Al) has the highest reflectance for UVC [30]. This means that aluminum parts can be used to increase the irradiation. This has been designed by [4] by integrating aluminum inside the enclosure where UVR is present.

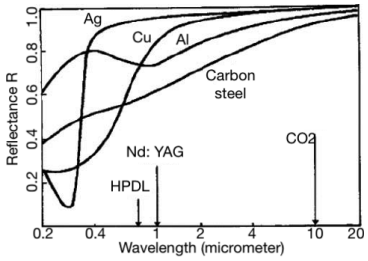
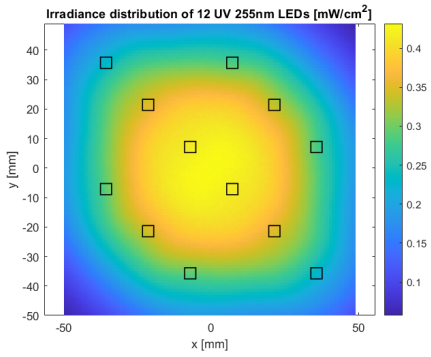


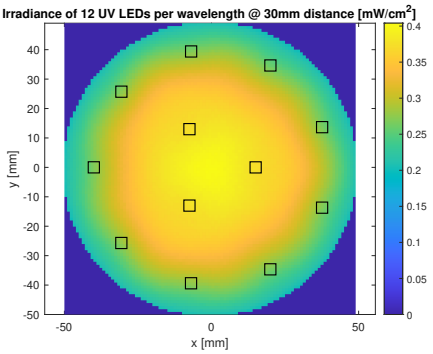
Figure 3.3: The reflectance spectrum of different metals [30].

3.5. LED placement for uniform irradiance

By placing the LEDs in a specific way, optimal uniform irradiance can be achieved. This depends on the amount of LEDs, but also the shape in which the LEDs are placed. Simulations ran by [5] demonstrate the difference in irradiance for a square shape and a circular shape for the plate (Figure 3.4).



(a) Relative irradiance distribution of 12 255nm LEDs on a square plate, diagonally placed [5].



(b) Intensity distribution of 12 255nm LEDs on a circular plate, with 2 rings with respective radii [15,40] cm and [3,9] LEDs per ring [5].

Figure 3.4: A side by side illustration of the irradiance of square and circular shapes

Figure 3.4 clearly shows that a circular shaped design provides a more uniform irradiance than the square shaped design. However, it is still not completely uniform. An optimization script has been implemented [4] which runs an algorithm multiple times finding the optimal placement of the LEDs. This results in the irradiance pattern found in Figure 3.5.

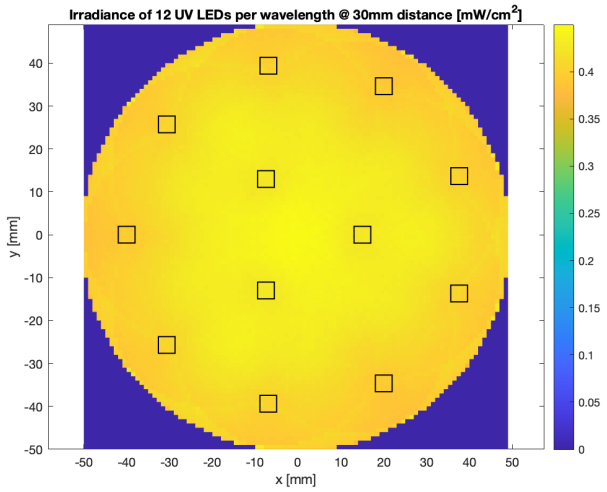


Figure 3.5: Intensity distribution of 12 255nm UVC LEDs on a circular plate, with 2 rings with respective radii [15,40] cm and [3,9] LEDs per ring, and a reflection coefficient of 0.75 [4], [5].

As mentioned in Section 3.2, there are a total of three LEDs that need to be packaged onto this panel. This can be done by making use of Equation (3.3) [4]:

$$\begin{aligned} \theta_{inner} &= \tfrac{2}{3}\pi \cdot k_{inner} + \tfrac{2}{9}\pi \cdot l_{inner} \qquad & k_{inner} &= [0,1,2], \quad l_{inner} = [0,1,2] \\ \theta_{outer} &= (\pi + \tfrac{2}{9}\pi) \cdot k_{outer} & k_{outer} &= [0,1,\ldots,7,8] \end{aligned} \tag{3.3}$$

Equation (3.3) calculates the coordinates of the LEDs in each ring for each wavelength by taking into account that the LEDs must not clash with each other once placed onto the PCB. This results in the final placement design, as can be seen in Figure 3.6. Here it can be seen that there is a uniform irradiation on all places of the circular plate. Furthermore, there are two major problems of disinfecting Alternaria with UVC LEDs as described by [31]. Firstly, the construction of a dense area of LEDs. Secondly, achieving uniform irradiation over the area that needs to be decontaminated. The solution presented in this section has a big chance of solving this problem.

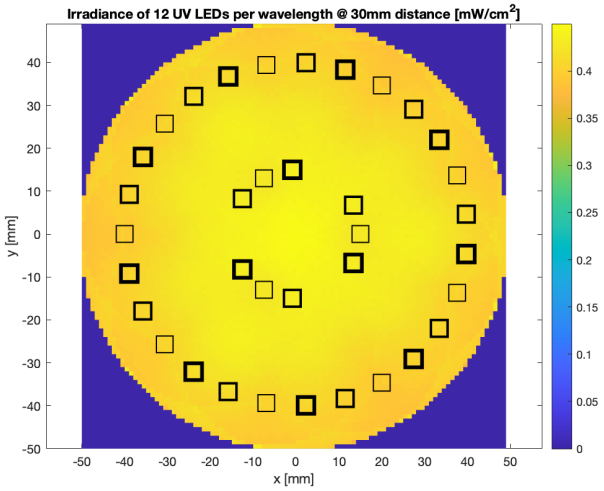


Figure 3.6: Intensity distribution of 36 UVC LEDs on a circular plate, with 2 rings with respective radii [15,40] cm and [3,9] LEDs per ring, and a reflection coefficient of 0.75 [4], [5].

3.6. Moving the seeds for uniform irradiance

One of the main requirements is Requirement [**SM.3.1**], where it is stated that the seeds must be irradiated on all sides. If the seeds are stationary, only the top and bottom will be irradiated. This means that some sort of mechanism must be implemented which makes the seeds rotate. This problem has been solved by [5] by implementing a motor that is connected to the circular quartz plate, with a mass imbalance attached to its shaft. Due to this imbalance, the motor will create vibrations [32]. This vibrational force that is induced by the motor is transferred to the quartz plate via four load carrying rods. This force will cause the seeds to rotate. This satisfies Requirement [**SM.3.1**], as in this way all sides will be irradiated.

3.7. Overview of the radiation design

All the design parameters in Sections 3.2, 3.3, 3.5 and 3.6 are implemented in one design, which can be seen in Figure 3.7. The LED panel (1) and the aluminum panel (2) can be attached onto each other. The quartz plate is placed in the middle. The stack of these two panels is also placed on the bottom side. At the bottom, the motor (4) can be found, with the imbalance mass connected to the rotary shaft. Through the attachment arms, the motor is connected to the casing of the quartz plate and induces vibrations to the seeds.

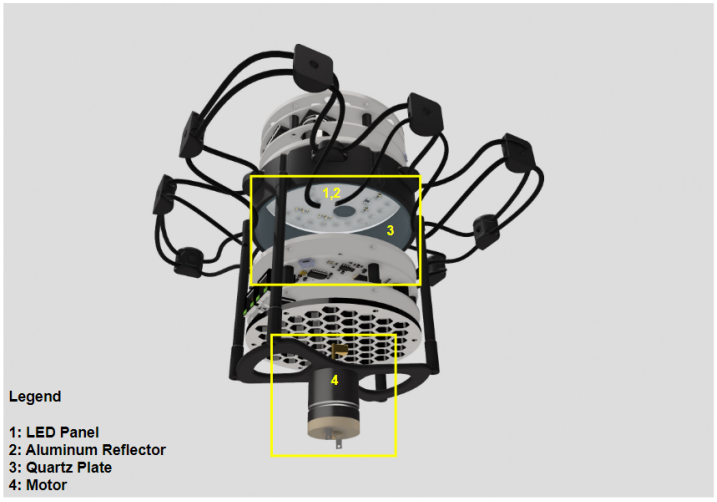


Figure 3.7: A breakdown of components of the radiation element of the device.

4

System Enabling

4.1. Introduction

According to Requirements [**SM.7**] and [**MM.4**], the Control Unit must only enable the system when the system is error-free. This requires the Control Unit to have an error detection protocol in either hardware logic or software. Due to the robustness of hardware logic and the fact that error detection is a critical part of a system that wants to operate reliably and safely, it was decided that a hardware error detection logic circuit should be designed on the Control Unit. This error detection circuit will also function as an enabling circuit such that if all checks have passed this circuit, only then can the other modules be activated. However, in case the hardware circuits fails, a software error detection functionality is also designed Sections 4.2.3 and 7.2.5

The other modules are equipped with adequate sensors to monitor temperature and currents for example. These are compared to a threshold value and exceeding that threshold, will trigger an error. The output of these comparators are called error flags. Figure 4.1 presents a block-diagram of this interface.

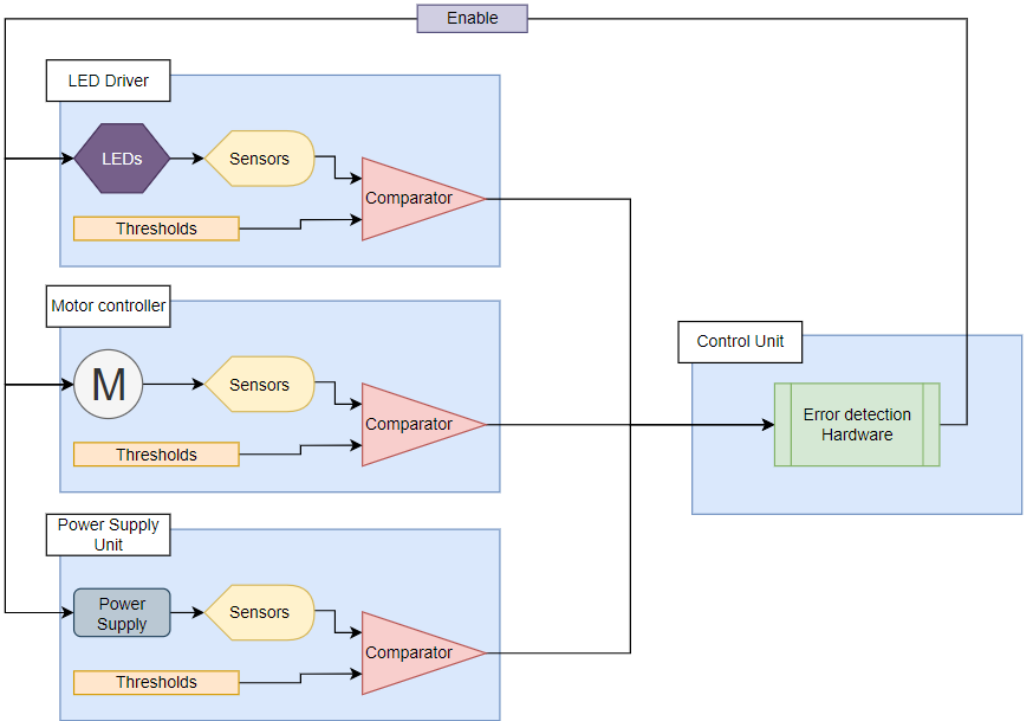


Figure 4.1: This overview illustrates how the other modules check for threshold violations and how the error detection hardware enables the modules.

4.2. Hardware error detection logic

4.2.1. Requirements

As mentioned before, this circuit is created mainly because of Requirements [**SM.7**] and [**MM.4**]. These requirements imply that all errors state should be inactive and must be compared. Thus the enable output signal should only be send to the other modules when there is no error. Considering the subrequirement [**MM.4.1**] it is stated that the error states can only be externally reset and not automatically by the Control Unit. This implies that a latching mechanism must be included in the circuit, that can be reset via an externally reset input.

4.2.2. Design of the logic circuit

This circuit must check all values of the error flags and if and only if all flags are in no error mode. A simple way to integrate this in a hardware design is with the using of simple logic gates, such as AND gates. The Boolean expression for an AND gate is $Y = AB$, meaning that the output Y shall only be high (1) when both inputs A and B are high (1). Any of the two being low (0) means that the output Y is always low (0).

In total there are five error flags that the Control Unit receives as an input:

- LED Driver Overcurrent Flag (OC_LED_FLAG); when the current draw by the LEDs is higher than the allowed maximum.
- LED Driver Overozone Flag (OO_LED_FLAG); when the ozone level exceeds the value defined by [**SM.7.3**].
- LED Driver Overtemperature Flag (OT_LED_FLAG); when the temperature of the LEDs exceeds their maximum allowed operating temperature.
- Motor Controller Overcurrent Flag (OC_MC_FLAG); when the current draw by the motor exceeds the maximum allowed current threshold.
- Power Supply Unit (OC_PS_FLAG); when the total current draw of the system exceeds the maximum allowed current threshold.

A simple way to solve this is to AND every single flag with each other. This means that it is required that the flags are active low, meaning that their value is a zero (0) when the threshold is exceeded. This way the output of the AND gates can be directly used to enable the other modules with an active high enable signal. A schematic of the circuit is displayed in Figure 4.2.

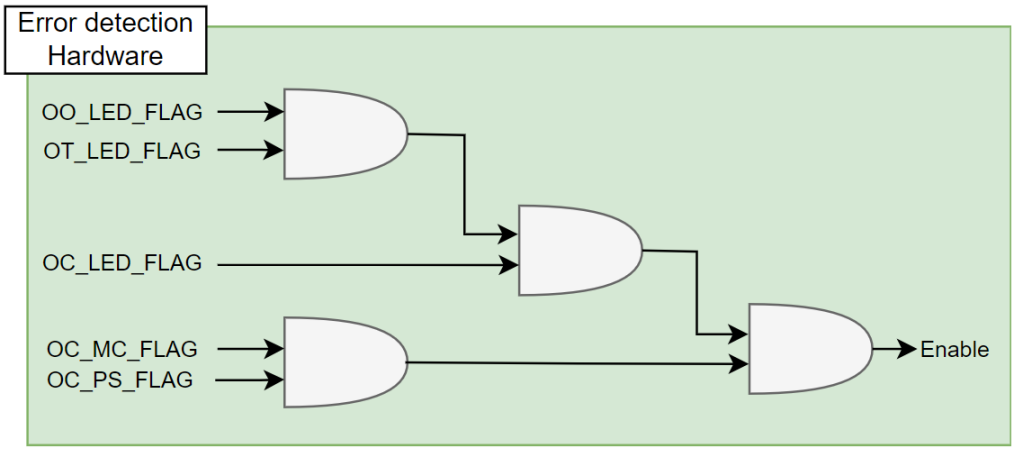


Figure 4.2: A schematic drawing of the error detection hardware logic.

A Boolean expression of this total circuit is given as:

$$Enable = OO\_LED\_FLAG \cdot OT\_LED\_FLAG \cdot OC\_LED\_FLAG \cdot OC\_MC\_FLAG \cdot OC\_PS\_FLAG \quad (4.1)$$

As it is required that the error states are resettable, some sort of latching must be applied. A latching mechanism that is equipped with a reset is a SR-latch [33]. Thus, before entering the circuit of the AND gates, the flags pass through SR-latches (NAND configuration) that latch their values.

To take preventive measures in case the latches did not work as intended, 0Ω do-not-place resistors have been placed between the input and output of the SR-latches. These can be placed during debugging of the system, in case the latches are preventing further progression in testing or integration.

4.2.3. Software bypass

In case that the hardware circuit has problems that could have happened during CAD or manufacturing, a back-up software error detection protocol can be implemented. This can be seen as a software bypass.

This can be implemented by retrieving these error flags from the other modules via the communication protocol. Furthermore, a command can be created that executes the above mentioned Boolean expression (Equation (4.1)). Commanding the microcontroller to pull one of the GPIO pins high whenever the output of Equation (4.1) is true (1) creates a +3.3V signal that can be labelled as Enable_SW, which is the software enable. With the addition of this software enable, the hardware must implement a feature that makes it possible for the software signal to bypass the circuit given in Figure 4.2. Stating that either the hardware enable or software enable has to be high (1), for the enable signal to activate requires the implementation of an OR gate. This can be seen in Figure 4.3. A schematic of the implementation in the PCB design can be seen in Appendix A.

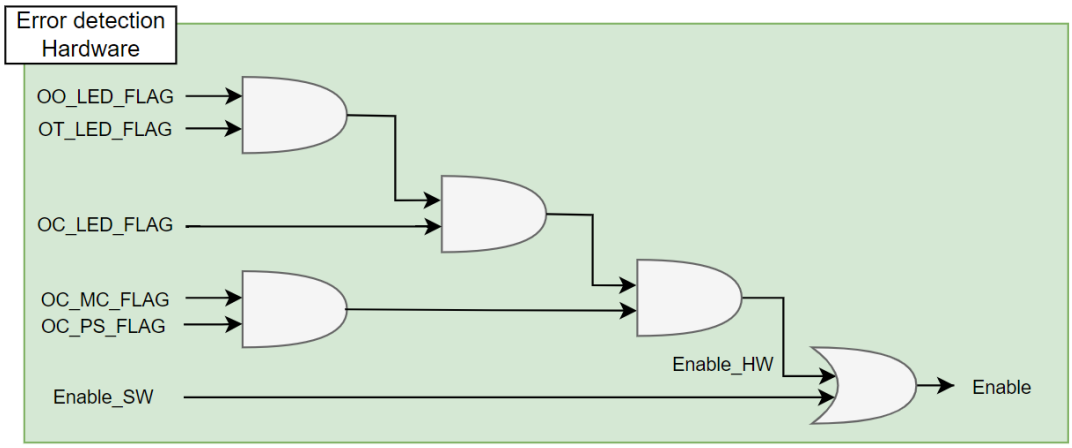


Figure 4.3: A schematic drawing of the error detection hardware logic with the software enable bypass implemented.

The Boolean expression in Equation (4.1) can be adapted to the following with the addition of the bypass:

$$Enable = Enable\_HW + Enable\_SW \tag{4.2}$$

5

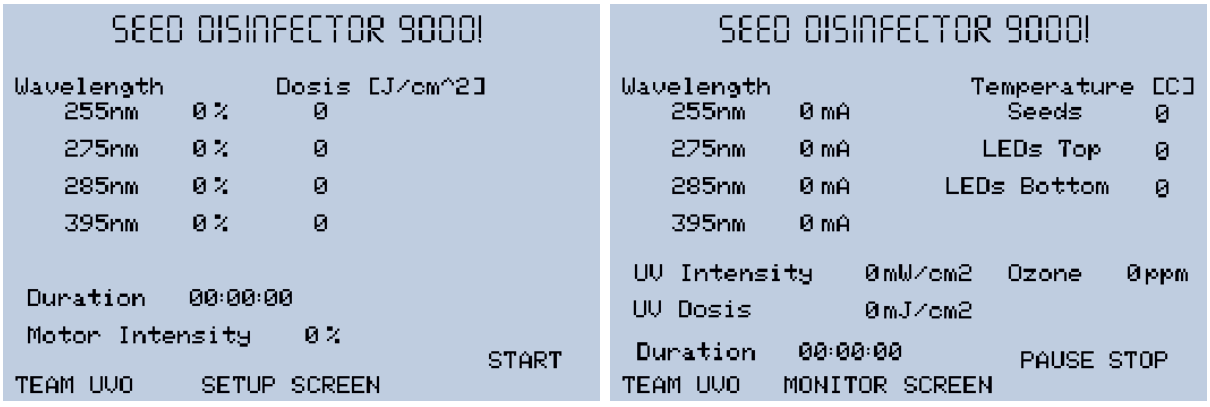
Implementation of a User Interface

5.1. Introduction

The Control Unit requires a User Interface (UI) to be able to display measurement data and to change the radiation pattern, as per Requirement [**MM.5**] (and thereby [**SM.8.1**] and [**SM.9**]). The requirement of the ability to change parameters ([**MM.5.2**]) exists due to clashing literature, as explained in Section 3.2. The flexibility of the device parameters allows for testing of multiple setups to determine the optimal radiation pattern. Displaying real-time data must be implemented for user feedback from the system ([**MM.5.1**]). In addition to this, the UI can also be used to raise errors ([**MM.5.4**]), and to solve said errors ([**MM.4.1**]).

5.2. Displaying on Screen using GUISlice

The screen implemented is a thin-film-transistor (TFT) liquid-crystal display (LCD). It is 3.5 inches (8.89 cm) and 480 by 320 pixels. Therefore, the design of the menu must fit within those boundaries. A library called GUISlice [35] and its development environment GUISLiceBuilder [36] are used. The following menu screens are created: a setup screen and a monitor screen. These can be seen in Figure 5.1. On these menus, values are presented to the user. However, refreshing the screen often results in flickering. To resolve this issue, only the elements that have changed get redrawn. To import the menus into the software Listing E.1 was developed.



(a) The preview of the Setup Menu Screen

(b) The preview of the Monitoring Screen

Figure 5.1: The two default screens the system switches between.

5.2.1. Setup Screen

Figure 5.1(a) displays the setup menu for the device. The design is kept simple so that it is easy and intuitive for the user. The elements that can be changed are the relative intensities for every wavelength, the duration and the motor speed. Therefore, fulfilling Requirements [**SM.8.1.1**] through [**SM.8.1.3**].

On the right there is an array for the dose. This dose is the expected one per wavelength, which is calculated based on an estimation of the relative intensity multiplied by the duration. On the bottom right, there is a start button to commence the disinfection procedure.

5.2.2. Monitor Screen

When the system is in the process of disinfecting, Figure 5.1(b) is shown on the display. The current measurements per wavelength are shown. In addition to this, there is only one UV intensity measurement. Merely one UV sensor is installed, which measures a spectrum of wavelengths. As a consequence, the intensity can only be measured as a combination of the individual wavelengths. To compensate, the current measurements of every wavelength are shown, in order to indicate the relative intensity. The total absorbed dose is also calculated as a summation of the intensities over time. Lastly, the temperatures, ozone concentration and the remaining duration are visible.

5.2.3. Error Popup Screen

Should something go wrong with the system, it of course must shut down. In that case, the user must be notified. Therefore, an Error Popup Screen is created as shown in Figure 5.2. The default error messages can be replaced by a more descriptive indication using the received errors (Section 6.4.2). The continue button is there to ensure user interaction, as specified in [**MM.5.4**]. Were this requirement not in place, the Control Unit could reset itself without the need of a human. If done incorrectly, it could lead to enabling the defect system, ultimately damaging the device and risking the safety of its environment.

Figure 5.2: The default error message screen

5.2.4. Physical User Input

The physical user input consists of a rotary encoder with button functionality and two additional stand-alone buttons (Figure 5.3). Simplicity is key, both for user experience and implementation difficulty. This design is inspired by an oscilloscope, which is (probably) familiar to the user. It also saves pins on the microcontroller by as little buttons as possible. The rotary encoder acts as a turning knob. This results in a total of one turning knob and three buttons. This is enough for an “up”, “down”, and “enter” button (the rotary button) and a knob.

The rotary encoder resembles a potentiometer in shape, but behaves differently. It consists of a disc with sectors and probes at two stationary points. When rotating the encoder, the direction can be determined by examining the order in which the probes make contact with the sectors. The software implementation of the buttons and rotary encoder are done by the Button2 [37] and ESP Rotary [38] libraries.

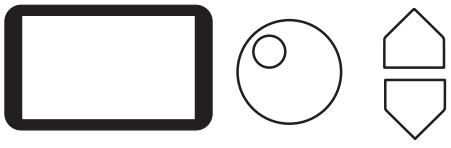
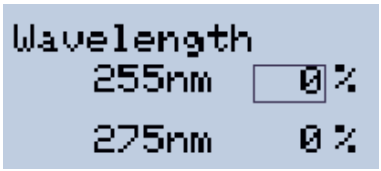


Figure 5.3: Representation of the layout of the User Interface. This schematic is not to scale. From left to right: the screen, the rotary encoder and the up and down buttons.

5.2.5. Visual feedback of the User Input

The user needs to be informed which element is currently selected, or whether the user is editing a value. The user can move the cursor using the up and down buttons. The cursor is displayed using a box (Figure 5.4(a)). If the user presses the “enter” button, the value can be edited using the rotary knob. When editing, the background color changes and the text becomes white (Figure 5.4(b)). This high contrast alerts the user. This decision is based upon everyday examples. When placing your mouse cursor, it also becomes a line. When selecting something, the mouse highlights the text, as in this design.



(a) The box surrounding an element indicates that the element is selected.

(b) To indicate that the user is currently editing an element, its background color and text color change for a bigger contrast.

Figure 5.4: Visual feedback of the User Input

5.3. LED Display

There is a LED display implemented onto the PCB, which are basically some LEDs that turn on or off depending on the state of the signal driving the LED. For example, for the error flags, red LEDs are used that turn on when an error is detected. The presence of voltage is also indicated by green LEDs that turn on when power is on. Furthermore, three yellow LEDs turn on whenever a reset signal is sent to the other modules. Finally, a green LED turns on whenever the error detection hardware detects no error. This LED is connected to the enable signal via N-Channel MOSFET. A schematic of this display can be found in Appendix A.

6

Communication Protocol

6.1. Introduction

The system consists of multiple modules, with the Control Unit in charge. In order to control the modules ([**SM.7**], [**MM.1**]) and request information ([**SM.9**], [**MM.5**]), there must be communication between the modules ([**SM.5**] and [**MM.2**]). This chapter explains the considerations and workings of the developed communication protocol. Using the developed communication protocol, sensor values can be requested and drivers can be set.

The communication protocol is built on top of an already existing communication protocol. In the remainder of this thesis, the already existing protocol is referred to as the base protocol. The communication protocol developed for the system is referred to as the communication protocol. In base protocols for communication, there is often a module which initiates, and one that responds. The prior is referred to as the controller. The latter is called the target. In order to avoid confusion, whenever the report mentions a controller, a controller in communication is meant. The Control Unit is always written out fully.

6.2. Selection of Base Protocol

There are multiple base protocols available for communication between different modules. Common options are I^2C (Inter-Integrated Circuit) [39], SPI (Serial Peripheral Interface) [40] and UART (Universal Asynchronous Receiver/Transmitter) [41]. However, each protocol has its own strengths and weaknesses. Table 6.1 shows a list of relevant properties.

Properties	**I^2C [39]**	**SPI [40]**	**UART [41]**
Number of controllers	1 or more	1	2*
Number of targets	1 or more	1 or more	2*
Maximal number of targets	128	As many chip select pins as you have available	-
Minimal number of wires	2	3 + 1 Chip Select	2
Maximum number of wires	2	3 + 1 per target	2
Typical Data Rate	100 kbits/s	50 Mbit/s	9600 bit/s

Table 6.1: I^2C is best used for communication in a complex system with multiple controllers and targets. SPI is best used for connecting interfaces. *UART can only be implemented between two modules, where both function as controller and target.

6.2.1. UART

UART can only be used for connecting two modules together. As the Control Unit requires communication with three other modules, this protocol is not suitable. If implemented, UART would require the Control Unit PCB to have three UART chips. This is cost inefficient and illogical to do, as this protocol is

simply not designed for this specific implementation.

6.2.2. SPI

SPI has a lot of advantages. It allows for high data rates and can implement as many targets as needed. However, the cost of every target is another chip select connection. Due to the limited number of pins on the microcontroller, it is unwise to dedicate so many pins to communication. In addition, the protocol supports only one controller. Therefore, no modules can initialize communication except the Control Unit. Since it is plausible that the other modules could also want to request or send information from or to the Control Unit, SPI is not suited for this use.

6.2.3. I^2C

I^2C , as the name suggests, is best used between multiple different ICs. Due to its expandability for both the number of controllers and the number of targets, it is chosen as the basis for the overall inter-module communication design. Every module can both initialize communication with the Control Unit and respond to requests from the Control Unit. The fixed number of connections needed, allows for additional use cases of the microcontroller pins and the connectors.

6.3. Class Implementation

In the code, the Wire library [42] handles I^2C communication. Every system module initializes this library with their respective I^2C address (as listed in Listing E.7), hereby becoming part of the I^2C bus. In order to abstract the library functions, a light-weight wrapper class with methods to connect to the I^2C bus is created, called `I2CInterface` (Listings E.8 and E.26). This class is used inside the `CommunicationInterface` class (Listings E.9 and E.27), as shown in Figure 6.1. The Communication Interface calls functions from the I^2C Interface and receives answers. The rest of the software uses the `CommunicationInterface` class and does not know about the I^2C interface. This implementation has two big advantages. Firstly, the `CommunicationInterface` class does not directly handle the library functions. Should a different library be used, a new wrapper class can be created. The `I2CInterface` can then easily be swapped out, without having to rewrite the more complex `CommunicationInterface` class. The second advantage is reusability. The `I2CInterface` class can be used for the other module software as well.

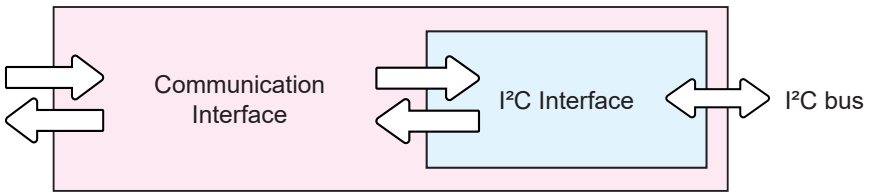


Figure 6.1: The I^2C protocol is wrapped inside `I2CInterface` and communicates with the I^2C bus. The `CommunicationInterface` is connected to the rest of the code.

6.4. Template of Protocol Messages

In order to structure the communication with other modules, a general outline for transmission is created (Figure 6.2). Every transmission consists of three parts: sending the command via a data package, a request and receiving the response. Splitting the command into multiple parts, the communication protocol becomes very flexible and expandable. A lot of different commands can be implemented this way. The software implementation of the communication protocol can be found in Listing E.7.

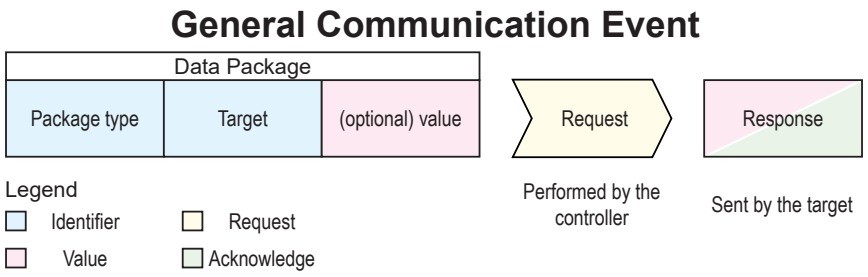


Figure 6.2: Blue, red, yellow, and green blocks respectively indicate identifiers, values, requests, and acknowledgements. Each block represents one byte.

6.4.1. Tokens

Every block in Figure 6.2 represents a token, a part of the message consisting of one byte. In the code, tokens are implemented using a `unsigned char`. Defining such a type in software, increases readability of the code. Every time a token is mentioned, it can be understood that communication is involved. In addition to this, tokens ease the effort needed to implement behavior. Because every package is split up into tokens, and because every token specifies one part of the instruction, the instruction can be decoded without the need of a long lookup table.

There are three types of tokens: identifier, value and acknowledgement tokens. (The reason "Request" is not a token is explained in Section 6.4.3.) Identifiers are there to describe what to do or who to target, acknowledgements give feedback to the controller and values convey information.

Certain special tokens are also defined in the communication protocol: the acknowledge (`ACK`), not acknowledge (`NACK`) and invalid (`INVALID`) tokens. The acknowledge and not acknowledge tokens ought to be received as a confirmation that the target has executed his command. For example, the Control Unit wishes to turn on the top LED driver. It could check if it worked by requesting sensor data, but a simple acknowledge is easier. The mindful reader might still wonder why it is necessary to implement such a system: I^2C already has built-in acknowledgements. That is correct. However, using that information, it can only be derived whether the target has successfully received the message. If the target is unable to execute the command, the controller is not informed. The other special token is the invalid token. All data should be initialized to the invalid token. If the invalid token is read, the controller or target knows that something has gone wrong in communication and can act accordingly. No other token is allowed to have this value, except for tokens which represent a value (red in Figure 6.2).

All Token definitions can be found in Appendix C.

6.4.2. Sending the Command

The command data package consists of two essential parts: the package identifier token and the target identifier token. The package token displays the type of the command. Currently, there are four package type tokens implemented, as can be seen in Figure 6.3. These are: request sensor data, set driver intensity, set variable resistor and error.

The need for the "Request Sensor Data" and "Set Driver Intensity" tokens is self-explanatory. With the former, sensor data can be requested for data logging. With the latter, the individual led drivers and the motor driver can be controlled. The third token is created because the LED driver has variable resistors [4]. Some of these digital resistors are part of feedback loops in the sensors. So the ability to set the values of the resistors, permits the implementation of user specified sensitivities per sensor. The last package type, the "Error" package, is received by the Control Unit. Since the error flags are not present at the input of the microcontroller (Section 4.2.3), they are to be transferred via I^2C . The Control Unit can act accordingly when receiving an error.

Then the target identifier is specified: which sensor data should be read out? What driver intensity should be altered? Every system module has a list of its possible targets. By specifying a list per possible

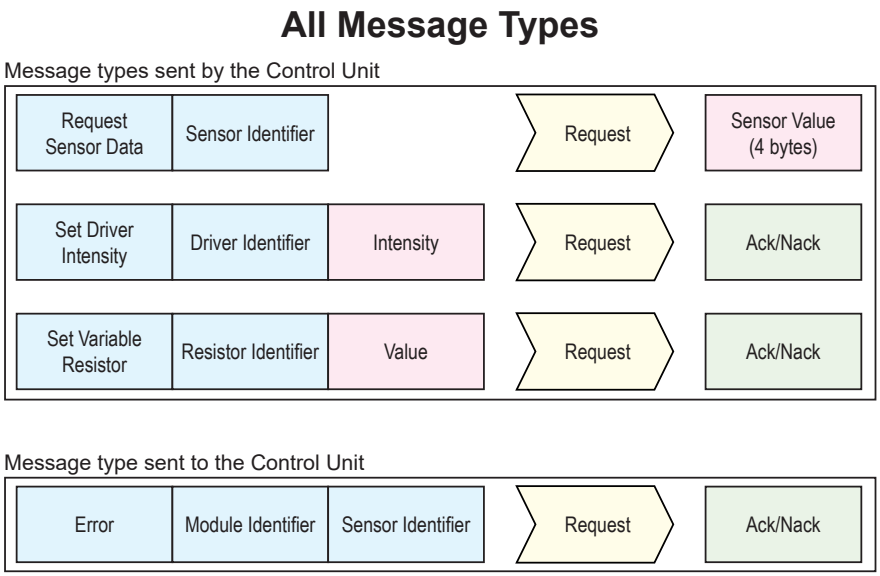


Figure 6.3: A schematic overview of all message types.

package token, up to 255 targets can be implemented for every command! In the case of an "Error" package, the identifier is the I²C address of the controller[1]. Sending merely the sensor identifier is not sufficient, as identifier tokens are not unique to one system module. For example, `CURRENTSENSOR_255nm` and `CURRENTSENSOR_MOTOR` have the same value. Should the Control Unit receive such an error message, it would be unable to derive where the error originated. This is more likely to occur if more sensors are implemented.

The last token of the data package is dependent per command. It can be a value ("Set Driver Intensity" and "Set Variable Resistor") or another identifier ("Error"). This is specified as per the protocol.

6.4.3. Request

The request is not realized explicitly through a token which is sent. In fact, it is part of the Wire library [42] as `Wire.requestFrom`. Here, the number of bytes requested can be specified, depending on what is expected. As of now, only "Request Sensor Data" requests 4 bytes, which form a `float`. The other commands require an acknowledgement token, so just one byte. The target of the communication determines in the function `Wire.onRequest` what should be responded, using every previously sent tokens to build towards the response.

6.4.4. Receiving the Response

This functionality can easily be implemented using the `Wire.requestFrom` function. The return value is the number of bytes which have been returned. Then, that number of bytes can be read from the I²C buffer. The Communication Interface can then pass these values to whoever requested them.

[1]Reminder: controller means controller in communication, not the Control Unit.

7

Prototype Design and Implementation

7.1. Introduction

This chapter presents the integration of the designs presented in Chapters 3 to 6 into one design. This will be the Control Unit Prototype Design. Furthermore, other design choices that were selected for making the PCB design of the Control Unit functional are also explained in short detail, along with some minor software applications. Afterwards, the implementation of the Control Unit into the system will be presented. This is important as wrong interfacing with the rest of the modules will cause problems when integrating everything.

7.2. Prototype Design

Before being able to start designing the functionalities of the Control Unit concretely, the overall system overview of the Control Unit must be constructed, thinking about the following problems:

- Input voltage conversion to usable voltage for ICs
- Programming the microcontroller via USB
- Microcontroller must be able to write and read data
- Microcontroller must be resettable
- Due to Requirement [**MM.8**] the microcontroller shall be an ESP32-WROOVER (from now on referred to as ESP32)
- The Control Unit sends reset signals to the other modules [**MM.9**]

Now that these problems are also stated, the total overview of the Control Unit hardware can be presented along with the designs given in Chapters 3 to 6. This overview can be seen in Figure 7.1

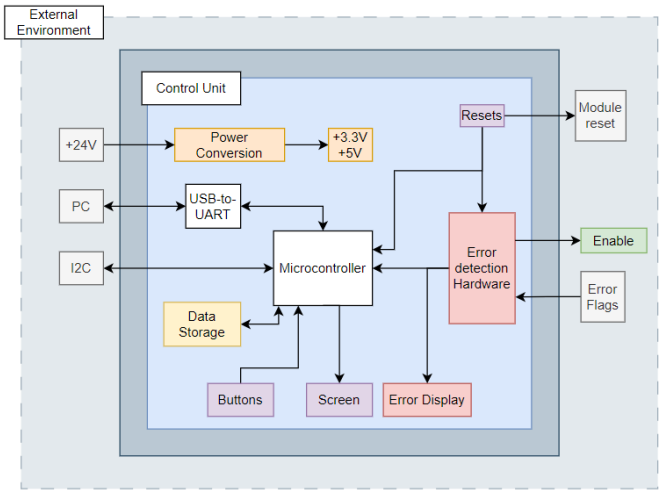


Figure 7.1: A block diagram presenting the system overview of the Control Unit.

7.2.1. General Software Information

The software is developed in C++1z (or also known as C++17) using the g++ compiler [43]. The ESP32 Arduino Core framework [44] is used, which is based on Arduino [45]. Therefore, the code for the Control Unit can be tested using Arduino UNOs, which are not based on the ESP32-WROVER [46]. The reason for this choice of language and framework, is because the Arduino framework is built for embedded applications and very easy to use. While C++11 is the default version, C++1z was chosen because of the introduction of the `inline` keyword. This allowed for easier implementation of the `sensor` structs in Listings E.10 and E.24. Now, the sensors can be defined in the header files without having to initialize them using a function.

7.2.2. External Connections

As can be seen in Figure 7.1, there are a total of five types of in-/outputs (*External Environment in Figure 7.1*). These are the error flags as described in Chapter 4, the communication via I^2C as per Chapter 6, receiving the input voltage from the power supply unit and independent reset signals for the LED drivers and motor controller. This requires the addition of connectors to the Control Unit. This problem has solved by implementing RJ45 headers [47] on the PCB such that Ethernet cables can be used for external communication and connections. The main reason for using this type of interconnecting is that it is easy to implement and does not require production of wires, because they can be bought off-the-shelf. Another benefit of implementing Ethernet cables is that separate cables or wires for data lines and power lines is not necessary. Being able to use only one cable, total packaging becomes an easier process. In total four headers need to be included in the design, one for each module: Top LED Driver, Bottom LED Driver, Motor Controller and the Power Supply Unit. A schematic can be found in Appendix A.

7.2.3. Power Conversion

The input voltage received from the power supply unit is 24 V, which is a voltage level that can not directly power most of the ICs. This voltage needs to be converted to lower voltages that fall in the range of the allowed input voltage of common ICs, which are 3.3 V and 5 V. These conversions are done by using two different voltage regulators ([48] for 5 V conversion and [49] for 3.3 V conversion). These regulators only require the addition of an input decoupling capacitor and output stabilizing capacitor, as per the datasheets [48], [49].

To make the Control Unit also receive power from the PC and not solely from the supply, a circuit was implemented which connects the USB BUS voltage (5 V) to the general 5 V line. The circuit and how it works is are both presented in Figure 7.2. The schematic design of the power conversion is given in Appendix A.

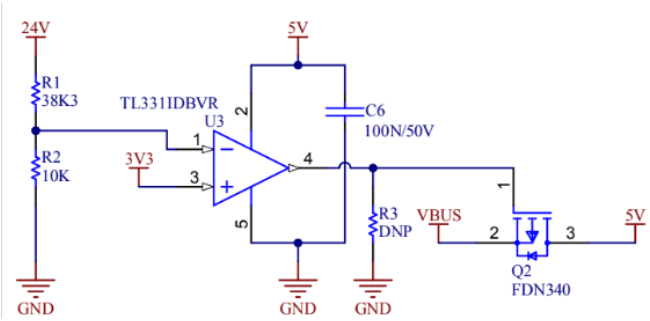


Figure 7.2: The comparator checks if the power supply is connected by comparing the 3.3 V threshold to the node voltage of the voltage divider. If the power supply is connected, the output of this comparator will be high, causing the P-Channel MOSFET (Q2) to be in a not conducting, thus not connecting VBUS to 5 V. This way, no current path from the power supply to the PC is formed. If the power supply is not connected, the gate of the MOSFET will be low, thus the MOSFET will be in conducting and powering the device via the PC.

7.2.4. USB Programmable

The ESP32 needs to be able to be programmable via USB. As the device itself is not directly a USB device [46], an interface needs to be implemented that the PC can recognize as an USB device and which can

conduct serial communication with the ESP32. Such an interface can be implemented via a simple USB-to-UART interface, such as the CP2102N-A02-GQFN28R [50]. By following the datasheets [46], [50], the signals and ports that need to be connected to get a properly functioning USB-to-UART interface can be made. This interface now makes it possible to program the ESP32 via USB, satisfying [**MM.7**]. A schematic of the interface and ESP32 is given in Appendix A.

7.2.5. Software Safety Measures

In addition to the hardware safety features, a software safety component has also been designed. When an error message is received by the communication interface (Chapter 6), the safety component will turn the enable signal off. Hereby disabling all modules. The error is decoded by determining from which module and sensor it was sent, using the communication protocol in Listing E.7. This error will then be displayed on the screen.

7.2.6. System State

In the software, the state of the system is stored in the system state (Listing E.6). The `SystemState` is a struct containing important information about the current state of the system. This list includes but is not limited to: the radiation parameters, the time the system has been running, and whether the system is running. But it will most definitely be expanded if the software receives an update.

7.2.7. Data Logging

The data logging has been designed such that it satisfies Requirement [**MM.6**]. A very simple and easy implementable way to store data is by using a SD card. This can be integrated onto the PCB by placing a SD cardholder. The SD card is connected to the ESP32 using SPI [40]. Luckily, the Arduino framework has a built-in SD card library [51]. This library carries out the creation of files and reading and writing to said files. A wrapper class can be designed in the same way that the `I2CInterface` is implemented, as explained in Section 6.3. The only concern of the implementation is ensuring that the chip select for the SD card is enabled and that the screen chip select is disabled.

7.2.8. Buttons

Buttons have been implemented for several functionalities. There are buttons to control the screen as mentioned in Chapter 5 but also to reset the Control Unit and the other modules. These are generic buttons that pull a signal high or low, depending on the configuration. The screen buttons and rotary encoder pull the signal high when pushed, and the reset buttons pull the signal low when pushed. Due to the renowned bouncing that happens in the voltage, a filter for debouncing [34] must be implemented. This can be done by placing a capacitor with a capacitance of 100 nF parallel to the button signal path. The design of the buttons can be found in Appendix A.

7.2.9. Voltage step-up

Most of the signals on the Control Unit have a high voltage level of 3.3 V, because that is the I/O voltage of the ESP32 [46]. However, the other modules operate on 5 V, meaning that a voltage step-up has to be implemented. The circuit and its explanation are presented in Figure 7.3.

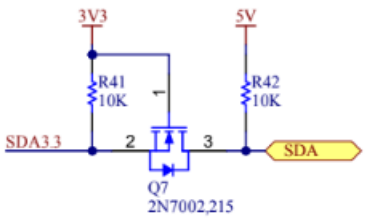


Figure 7.3: This circuit elevates the voltage level of a signal to a higher voltage. Whenever the signal at the source of the MOSFET is high, the N-Channel MOSFET will not conduct, thus the drain is being pulled high to a higher voltage. If the signal at the source is low, the MOSFET will conduct due to $V_{gs} > V_{Th}$, thus pulling the signal at the drain low. So one gets:
$V_s = 3.3V \Rightarrow V_{gs} < V_{Th} \Rightarrow V_d = 5V$ or $V_s = 0V \Rightarrow V_{gs} > V_{Th} \Rightarrow V_d = 0V$

7.2.10. Design for debugging and testing

Due to the Control Unit being a prototype and the short duration of the project, the Control Unit is designed in such a way that some features can easily be bypassed or that signals can be altered using pull-down, pull-up, 0 Ω and do-not-place resistors. For example, in case the hardware error detection does not work, and the software bypass also fails, the error flags can be manually pulled high via pull-up resistors. In this way, some systems not behaving as desired does not delay the total progress of the system. However, not all systems can be designed like thus, such as the USB-to-UART interface. This design philosophy satisfies Requirement [**MM.8**].

7.2.11. Final PCB Design for Control Unit

The final PCB model for the Control Unit is given in Figure 7.4. Schematics of the PCB can be examined in Appendix A.

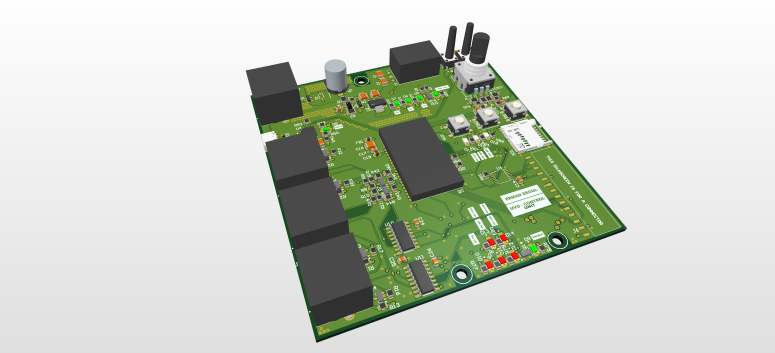


Figure 7.4: A capture of the final design of the Control Unit PCB.

7.2.12. Final Software Design for Control Unit

The overview of the software design for the Control Unit is given in Figure 7.5. Chapters 5 and 6 explain the User Interface and Communication in more detail.

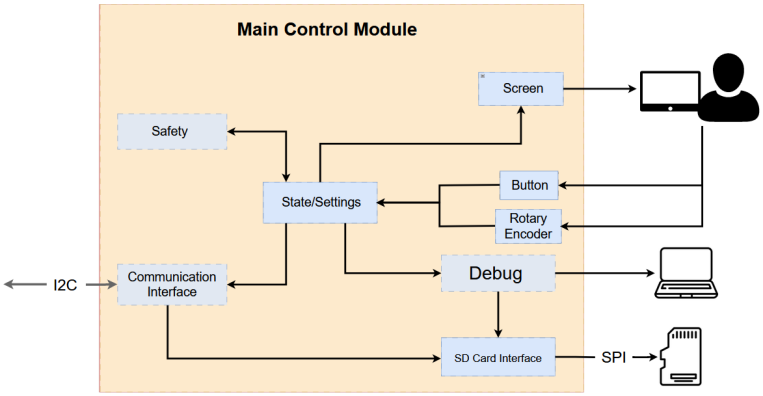


Figure 7.5: The simplified overview of the software modules.

7.3. Complete System Implementation

All other modules [4], [5] have also gone through the design process and the complete designed system is presented in Figure 7.6, along with a short elaboration on the interfaces. An assembled prototype can be seen in Figure 7.7, the PCBs are blank as these were inserted for fitting of the system and not for integration.

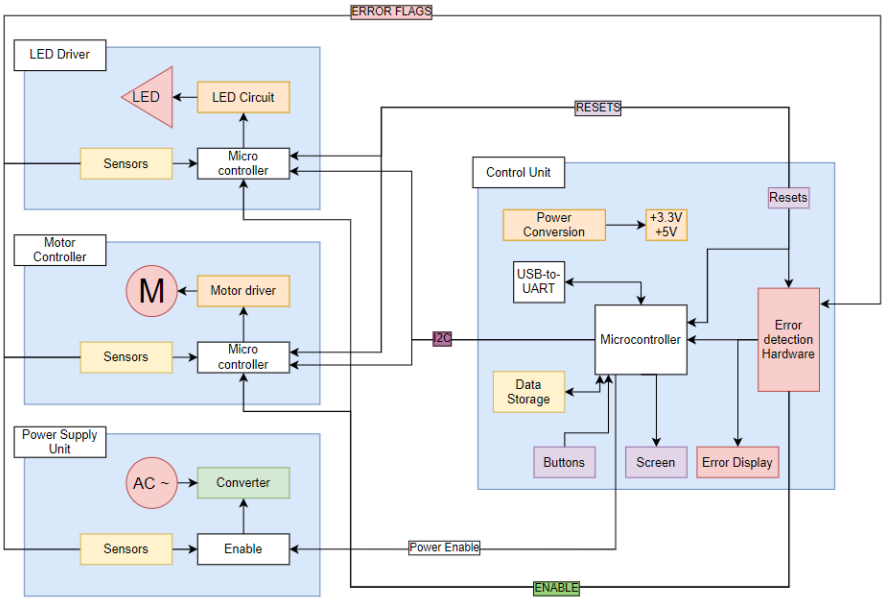


Figure 7.6: A complete overview of the system. The Control Unit communicates with the other modules via I^2C . The error flags are outputs of the sensory electronics located on the other modules and act as inputs for the error detection hardware. The Control Unit sends resets to the other systems to reset them. Enable is the output of the error detection hardware and enables the system, if no error is detected.

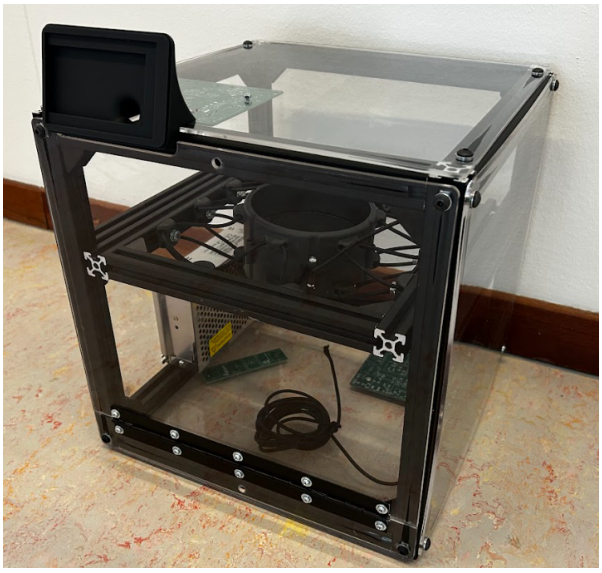


Figure 7.7: An assembled prototype of the complete system. PCBs are not assembled as this was built for fitting and not final integration.

8

Prototype Validation and Discussion of the Results

8.1. Powering up the PCB

The PCB has to be able to receive power from both the PC and from the power supply. This can be tested by plugging the PCB to either of the two. When plugging in the PCB to the computer via a USB cable, the PCB indeed powers up as all the power indicator LEDs turned on and the voltages were measured and all were correct. When using the power supply all voltages were measured and everything was fine. This means that the PCB can power up and that the power conversion circuits work as desired. However, when connecting the power supply, the LED for VBUS (power from USB) also turned on, which means the PC/Supply isolating circuit given in Figure 7.2 was faulty. During debugging it was noticed that the footprint of the PMOS was incorrect, meaning that it was always conducting. This can be fixed by soldering wires from the correct signal to the correct pins of the MOSFET, but was not done because this could for now be neglected. The fault does not cause problems downstream. However, it does result in a voltage drop of around 0.7 V, resulting in the 5 V line dropping to 4.3 V. This is only a problem when only using the power via USB.

8.2. Uploading to the PCB

For flashing the ESP32, the USB-to-UART circuit has to be tested first. This is a straightforward test as plugging in the USB cable onto the PCB would cause the VBUS LED to turn on, meaning that power was being received by the PCB. Furthermore, the PC could recognize the device as an USB device and allocated a COM port to the Control Unit. This meant that USB connection was succesful.

However, when trying to flash the code, no serial connection could be formed with the ESP32. After going through the schematics in Appendix A, it could be quickly detected that the TX and RX signals where not connected properly. The TX signal of the USB-to-UART IC should be connected to the RX pin of the ESP32 and vice versa. But the TX ports of both where connected to each other, which means there is no transmitter/receiver communication present. This was fixed by cutting the copper traces in the PCB and soldering wires to the correct ports.

This did, however, not solve all issues. It was found that the data terminal ready (DTR) port of the USB-to-UART was not connected to the DTR port of the ESP32. This was fixed by soldering a thin copper wire between the two.

After implementing these fixes to the PCB, it was possible to flash the ESP32 with code. However, this could only be done once, because after attempting to flash a later moment it did not work anymore. The error was that the serial data contained noise or corruption. This is a quite reasonable error as serial data lines are prone to interference, especially if they are fixed in such a fragile manner. If the PCB had a second iteration with the fixes mentioned in this section, this circuit would

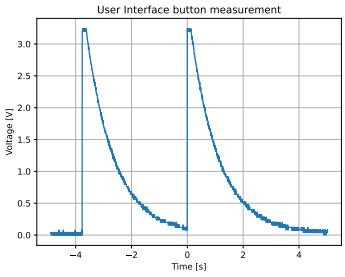
work without problems. A picture of the setup where the uploading is attempted can be seen in Figure B.1.

This issue was resolved by adding another PCB, that was not designed during this project. The microcontroller on that PCB is also an ESP32 and this means that it was possible to upload software on that PCB. This means that this issue is solved in the final system.

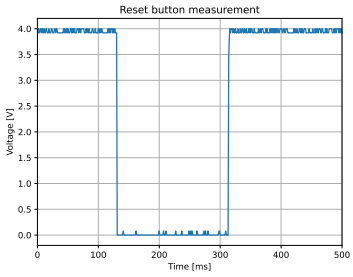
8.3. Testing of the User Inputs

8.3.1. Push buttons

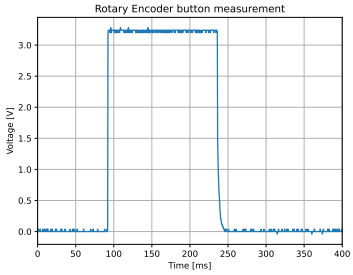
Firstly, the analog outputs of the buttons and rotary encoder were tested. That can be done by powering the PCB and using an oscilloscope. The oscilloscope ground should be connected to PCB ground and the signal probe. For the buttons that trigger an active state when going high, one would expect that the voltage level of the signal would jump to 3.3 V from 0 V and back. The buttons that have active low states, should do the opposite. The different buttons are: User Interface buttons (active high), reset buttons (active low), rotary push button (active high). The measurement results are shown in Figure 8.1.



(a) The measurement of the User Interface buttons, it can be seen that a discharge pattern can be recognized, because no pull-down resistor is implemented that causes a quick discharge.



(b) The measurement of the reset buttons, normal voltage level switching behaviour can be recognized. The reset button is active low. The peak voltage is around 4V, this is caused by the faulty footprint of the PMOS as mentioned in Section 8.1.



(c) The measurement of the rotary encoder push button, which demonstrates normal behaviour. The rotary encoder button is active high.

Figure 8.1: The measurements of the push buttons located on the Control Unit PCB.

In Figure 8.1(a), it can be seen that the User Interface buttons jump to 3.3 V, but slowly discharge back to 0 V. This can be related to the exponential discharge given by:

$$V(t) = V_0 \exp\left(\frac{-t}{RC}\right) \tag{8.1}$$

where

$V(t)$ = voltage of the signal path, V
V_0 = initial voltage of the signal path, V
t = time, s
R = resistance, Ω

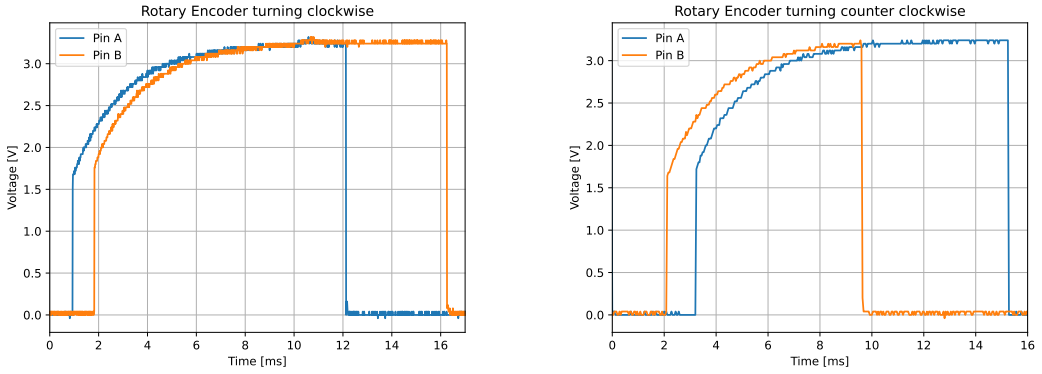
C = capacitance, F

Looking at the schematics Appendix A, it can be seen that there is no pull-down resistor from the signal line to ground, which means that the debounce capacitor gets charged, but has no quick discharge path which explains this behaviour. With the addition of a pull-down resistor, this behaviour can be solved. Although, this behaviour is present when pushing the button, it is not likely that it will cause problems when integrated with the User Interface.

The measurements for the reset buttons and rotary push button, given in Figure 8.1(b) and Figure 8.1(c), are as expected for buttons and can be confirmed to be functional.

8.3.2. Rotary encoder

The rotary encoder was also tested by probing the signal path with respect to ground using a oscilloscope. One would expect a behaviour as described in Section 5.2.4. At first the results were not correct. This was due to a pull-up resistor always pulling the signal path high and with an active high set-up this would not work. After changing the pull-up resistor to a pull-down resistor, the following results were obtained as illustrated in Figure 8.2. These are correct results. If one pin rises before another, a rotation in a certain direction can be derived.



(a) The voltage on pin A rises before pin B, indicating a clockwise turn.

(b) The voltage on pin B rises before pin A, indicating a counter clockwise turn.

Figure 8.2: Measurements of the rotary encoder

8.4. Testing the Error Detection Hardware

To test the error detection hardware, one must be able to reproduce two situations:

1. Being able to pull Enable high
2. Being able to pull Enable low

By applying an external voltage to each of the error flags (either a high voltage or a low voltage), these two situations can be reproduced. Firstly, by pulling all the flags high, the Enable LED indeed did turn on, indicating that Enable is high, which means that the system could be enabled as no error is detected. Pulling any of the flags to ground, triggered the error detection hardware and an error flag LED turned on and the Enable LED turned off, thus being able to disable the system. When applying a voltage to the exact same flag that previously had no voltage, still caused the error LED to be on and the Enable LED was still off. This indicated that the latching circuit is functioning as expected.

Although, the circuit works as expected, it can not be integrated in the system. The interface with the other modules had flaws. The other modules used active high for indicating an error, but the Control Unit uses active low. This means that this circuit can not work and the software must use the designed bypass function.

8.5. Testing the I²C communication

I²C is implemented in the software as per Chapter 6. The testing was done as presented in Figure B.3. This setup is representative of the entire system. The Arduino UNO makes use of the ATMega328p [52]. The same chip employed for the other module designs [4], [5]. An ESP32 is integrated in the Control Unit. However, the code can easily be compiled for this chip by altering some compile flags. This can be done by selecting a different environment in Listing E.2. For more information about the compilation, see Section 7.2.1. In the test setup, the code behaves as expected. Nonetheless, it remains unsure whether this is also the case for the Control Unit PCB. This has not been tested due to the uploading errors. In addition to this, the testing of the communication can only be performed when all PCBs are functioning.

8.6. Testing the Screen

Rik Imbens, one of our colleagues, has worked on a project including the same screen and an ESP32. This setup has been used to develop the screen functionality of the Control Unit. Figure B.2 demonstrates the working setup. A laptop is connected to the test setup via USB. It can be seen that the setup screen is correctly displayed on the test setup.

However, since the ESP32 was enclosed in the test setup, and the PCB was not working yet, the button functionalities could not be tested. Using code to simulate the buttons, it did show that the selecting and editing works. The screen can also display the parameters in the right place.

Figure B.1 shows the test setup for integrating the screen with the Control Unit PCB. This setup is not functional due to the uploading problems. When uploading works, this will be the setup to integrate the screen with the module.

The screen was not turning on completely, when powered by the Control Unit. Quickly, it was discovered that the footprint of the BJT-transistor that enables the screen was incorrect. Soldering some wires to the correct signals also did not fix the issue. The mistake there is expected to be a bad solder joint or a broken transistor due to the reworking process. By connecting this signal to ground via a 5.6 Ω, this issue can be fixed. This fix would mean that the screen would power up, without being regulated by the ESP32. Applying power to the Control Unit, also powers the screen immediately this way. However, after some more testing, it was found that the supply voltage and ground pins on the connector for the screen were flipped. This meant that the connector had to be fixed such that these two pins were now connected to the correct signals. After fixing this and using the external PCB for uploading software, the screen is ready to be used.

Furthermore, the implementation of the buttons and the user interface was also tested. Rotating the buttons and pressing the buttons did initiate an action on the screen. The user can select different parameters and adjust their values and start a process.

8.7. Testing the entire software

While developing, the code had to be rewritten to GUISlice (Section 5.2). Therefore, a lot of time was lost developing and debugging code that eventually did not get used. As explained in Section 8.2, there were problems with uploading. But there was no out-of-the-box ready hardware to act as a surrogate for the PCB to design the software on. In the end, only hardware was acquired to develop the screen and I²C software on. The combination of reworking a lot of code and little hardware to test on, resulted in the safety, debug, and data logging components which only have been designed, but not yet implemented in software nor tested.

9

Conclusions, Recommendations, and Future Work

9.1. Conclusions

To conclude, a PCB and software for the Control Unit Module is designed and tested. The PCB is designed to communicate with the other system modules, interact with the user via buttons and a screen, store data and have safety systems in the form of error latches. The error latches, buttons and flashing to the PCB have been tested. However, due to difficulties regarding uploading code to the PCB, the other functionalities could not be verified, since those require software to work.

A communication protocol via I^2C , its implementation and the Graphical User Interface is designed in software. These functionalities have been verified in testing environments, but not on the PCB due to problems with uploading.

The PCB has been completely debugged and all design mistakes and flaws were recovered. Nonetheless, it was possible to fix the problems to get the user interface to function as it should.

The evaluations of the Programme of Requirement (Chapter 2) are shown in Tables 9.1 and 9.2. A dash indicates that the requirement is not discussed anywhere. If a requirement is denoted as “tested”, the results are discussed Chapter 8.

The conclusion of disinfecting seeds can be found in the Test Report, which is included in Appendix F. In the end, all the modules together managed to build a device that is capable of exposing seeds to high amounts of Ultraviolet C radiation. The doses of the radiation can be externally specified by the user. The seeds get radiated from both sides and due to the vibrations induced by the motor, all sides will be radiated. The safety of the system is guaranteed by the error detection hardware. The development of such a device opens the door for lots of research and testing in the field of sustainable seed disinfection. The hypothesis that seeds can be disinfected using UVC has not yet been proven in this thesis Appendix F.

Table 9.1: Evaluation of the System Requirements

Requirement	Discussed where	Achieved
[**SM.1**]	Section 1.1	Designed and tested
[**SM.2**]	Section 1.1	Designed, tested but inconclusive results Appendix F.
[**SM.3**]	Chapter 3	Designed, but not tested
[**SM.4**]	Chapter 7, [4], [5]	Designed and tested
[**SM.5**]	Chapter 6,[4], [5]	Designed, but not tested as a whole
[**SM.6**]	Chapter 4	Designed and tested
[**SM.7**]	Chapter 4	Designed and tested
[**SM.8**]	Chapter 5	Designed and tested
[**SM.9**]	Chapters 5 and 6	Designed, but communication not yet tested between the modules
[**ST.1**]	[5]	Designed, but not measured
[**ST.2**]	[5]	Yes
[**ST.3**]	-	No
[**ST.4**]	-	No
[**ST.5**]	Chapters 4 and 6	Designed, but communication not tested between the modules
[**ST.6**]	-	No

Table 9.2: Evaluation of the Module Requirements

Requirement	Discussed where	Achieved
[**MM.1**]	Chapter 7	Designed and tested with motor controller
[**MM.2**]	Chapter 6	Designed and tested, but not on the PCB
[**MM.3**]	Chapter 4	Designed and tested
[**MM.4**]	Chapter 4	Designed and tested
[**MM.5**]	Chapter 5	Designed and tested
[**MM.6**]	Chapter 7	Designed, not yet implemented nor tested
[**MM.7**]	Chapter 7	Designed and tested, fixed with external PCB
[**MM.8**]	Chapter 7	Designed and experienced
[**MM.9**]	Chapter 7	Designed and tested
[**MT.1**]	-	No
[**MT.2**]	Section 7.2.12	Debatable, buttons are reachable by user, but uncomfortable.
[**MT.3**]	Chapter 7	Designed, not yet implemented nor tested
[**MT.4**]	-	No
[**MT.5**]	-	No
[**MT.6**]	Chapter 7	Implementation of Ethernet ports and Screen connector

9.2. Recommendations

As a suggestion for the hardware, crucial signals and circuits should be verified by another person than the designer. Crucial here refers to elements which cannot be bypassed with 0 Ω or DNP resistors. This way, mistakes such as incorrectly wiring the RX and TX lines would not have happened. In addition to this, should time allow it, a second version of the PCB should be made in order to correct mistakes.

For the software, it is strongly recommended to buy pre-existing hardware modules. These modules with tested behavior can be used to develop all software components. Then the development becomes far more efficient. It eliminates the dependency of software and hardware. Secondly, a detailed software layout should be created first. If all functionalities and dependencies are determined beforehand, big overhauls of the code layout are less likely to occur. When creating this layout, it is advisable to look for libraries which implement the functionalities you require.

Lastly, a more general proposal when creating such a design. The thesis can be written in parallel with the development. Less time has to be spent in the end, if every week a chapter is written for the thesis.

9.3. Future Work

In the future, more types of pathogen and seed can be studied experimented with in the machine. The strength of the machine is its scale of settings that can be changed. Since the machine is built for flexibility, it could be optimized a lot by omitting redundant components. Then, if the device is fully tested, it can be commercialized and enter the market. However, it should comply with existing safety regulations. To make it a fully cultivated stand-alone product, touchscreen, parameter presets and automatic seed loading and unloading should be installed.

Literature

[1] Rijk Zwaan Zaadteelt en Zaadhandel B.V. "Rijk Zwaan company website." (), [Online]. Available: https://www.rijkzwaan.nl/.

[2] G. Lenssen, *Welkom bij rijk zwaan*, Company Visit, 2023.

[3] H. van Zeijl, *Ultraviolet sterilization: Uvc seeds bacteria inactivation*, Graduation Project Proposal, 2023.

[4] R. Imbens and L. Klootwijk, "UVC LED driving and sensing unit, UVC Sterilisation," Thesis, 2023, p. 50.

[5] D. Schat and M. Mazurovs, "UVC Sterilization, Mechanics Group, EE3L11: Bachelor End Project," Thesis, 2023, p. 50.

[6] S. J. Balk, the Council on Environmental Health, and S. on Dermatology, "Ultraviolet Radiation: A Hazard to Children and Adolescents," *Pediatrics*, vol. 127, no. 3, e791–e817, Mar. 2011, ISSN: 0031-4005. DOI: 10.1542/peds.2010-3502. eprint: https://publications.aap.org/pediatrics/article-pdf/127/3/e791/907919/zpe0031100e791.pdf. [Online]. Available: https://doi.org/10.1542/peds.2010-3502.

[7] Canadian Centre for Occupational Health and Safety. "Ultraviolet radiation." (2022), [Online]. Available: https://www.ccohs.ca/oshanswers/phys_agents/ultravioletradiation.html.

[8] UV Resources. "Why UV-C Cannot Produce Ozone." (2022), [Online]. Available: https://www.uvresources.com/the-ultraviolet-germicidal-irradiation-uv-c-wavelength/.

[9] V. Sharma and H. Demir, "Bright Future of Deep-Ultraviolet Photonics: Emerging UVC Chip-Scale Light-Source Technology Platforms, Benchmarking, Challenges, and Outlook for UV Disinfection," *ACS Photonics*, vol. 9, no. 5, pp. 1513–1521, Apr. 2022. DOI: https://doi.org/10.1021/acsphotonics.2c00041. [Online]. Available: https://pubs.acs.org/doi/10.1021/acsphotonics.2c00041.

[10] W. Kowalski, *Ultraviolet Germicidal Irradiation Handbook*, 1st ed. Berlin: Springer Berlin, Heidelberg, 2014. DOI: https://doi.org/10.1007/978-3-642-01999-9.

[11] N. Jiang, Z. Li, L. Wang, *et al.*, "Effects of ultraviolet-c treatment on growth and mycotoxin production by alternaria strains isolated from tomato fruits," *International Journal of Food Microbiology*, vol. 311, p. 108 333, 2019, ISSN: 0168-1605. DOI: https://doi.org/10.1016/j.ijfoodmicro.2019.108333. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168160519302636.

[12] S. Bates. "DEOXYRIBONUCLEIC ACID (DNA)," National Human Genome Research Institute. (Jun. 2023), [Online]. Available: https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid (visited on 06/16/2023).

[13] M. Raeiszadeh and B. Adeli, "A Critical Review on Ultraviolet Disinfection Systems against COVID-19 Outbreak: Applicability, Validation, and Safety Considerations," *ACS Photonics*, vol. 7, no. 11, pp. 2941–2951, Sep. 2020. DOI: https://doi.org/10.1021/acsphotonics.0c01245. [Online]. Available: https://pubs.acs.org/doi/full/10.1021/acsphotonics.0c01245.

[14] "Effects of Ultraviolet Light on the Eye: Role of Protective Glasses," *Environmental Health Perspectives*, vol. 96, pp. 177–184, Dec. 1991. DOI: https://doi.org/10.1289/ehp.9196177. [Online]. Available: https://ehp.niehs.nih.gov/doi/epdf/10.1289/ehp.9196177.

[15] Global Light Association, "UV-C SAFETY GUIDELINES," *Global Lighting Association*, 2020. [Online]. Available: https://www.globallightingassociation.org/images/files/publications/GLA_UV-C_Safety_Position_Statement.pdf.

[16] J. D'Orazio, S. Jarrett, A. Amaro-Ortiz, and T. Scott, "UV Radiation and the Skin," *Radiation Toxicity in Cells*, vol. 14, no. 6, pp. 12 222–12 248, Jun. 2013. DOI: https://doi.org/10.3390/ijms140612222. [Online]. Available: https://www.mdpi.com/1422-0067/14/6/12222.

[17] A. Bhattacharyya, "The detrimental effects of progression of retinal degeneration in the visual cortex.," *Frontiers in cellular neuroscience*, vol. 16, Jul. 2022. DOI: 10.3389/fncel.2022.904175. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9372284/#:~:text=During%20retinal%20degeneration%20(RD)%20the,faces%2C%20read%20and%20find%20objects..

[18] P. H. Gies, C. R. Roy, S. Toomey, and A. McLennan, "Protection against solar ultraviolet radiation," *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, vol. 422, no. 1, pp. 15–22, 1998, ISSN: 0027-5107. DOI: https://doi.org/10.1016/S0027-5107(98)00181-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002751079800181X.

[19] I. Lager, V. Scholten, E. Bol, C. Richie, and S. Izadkhast, *Bachelor graduation project*, Bachelor Graduation Project Electrical Engineering, 2023.

[20] N. Okamoto, J. Hidema, and A. Higashitani, "222 nm far-UVC efficiently introduces nerve damage in Caenorhabditis elegans," *Plos One*, Jan. 2023. DOI: https://doi.org/10.1371/journal.pone.0281162. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0281162.

[21] C. Green and P. Scarpino, "The use of ultraviolet germicidal irradiation (UVGI) in disinfection of airborne bacteria.," *Environmental Engineering and Policy*, vol. 3, pp. 101–107, Dec. 2001. DOI: https://doi.org/10.1007/s100220100046. [Online]. Available: https://link.springer.com/article/10.1007/s100220100046#citeas.

[22] F. Palma, G. Baldelli, G. Schiavano, G. Amagliani, M. Aliano, and G. Brandi, "Use of Eco-Friendly UV-C LEDs for Indoor Environment Sanitization: A Narrative Review," *Atmosphere*, vol. 13, no. 9, p. 1411, Jul. 2022. DOI: https://doi.org/10.3390/atmos13091411. [Online]. Available: https://www.mdpi.com/2073-4433/13/9/1411.

[23] C. Ortiz-Mateos, "Best wavelengths for disinfection in the age of Sars-CoV-2 (corona-virus)," Article, Jul. 2020, p. 9. [Online]. Available: https://phoseon.com/wp-content/uploads/2020/07/Best-UV-wavelengths-for-disinfection-in-the-age-of-coronavirus-final.pdf.

[24] R. Kamel, M. El-kholy, N. Tolba, A. Amer, A. Eltarawy, and L. Ali, "Influence of germicidal ultraviolet radiation UV-C on the quality of Apiaceae spices seeds," *Chem. Biol. Technol. Agric.*, vol. 9, p. 89, 2022. [Online]. Available: https://link.springer.com/article/10.1186/s40538-022-00358-4#citeas.

[25] M. Begum, A. D. Hocking, and D. Miskelly, "Inactivation of food spoilage fungi by ultra violet (UVC) irradiation," *International Journal of Food Microbiology*, vol. 129, no. 1, pp. 74–77, 2009, ISSN: 0168-1605. DOI: https://doi.org/10.1016/j.ijfoodmicro.2008.11.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168160508006144.

[26] A. Kheyrandish, M. Mohseni, and F. Taghipour, "Development of a method for the characterization and operation of uv-led for water treatment," *Water Research*, vol. 122, pp. 570–579, 2017, ISSN: 0043-1354. DOI: https://doi.org/10.1016/j.watres.2017.06.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0043135417304906.

[27] K. Song, F. Taghipour, and M. Mohseni, "Microorganisms inactivation by wavelength combinations of ultraviolet light-emitting diodes (uv-leds)," *Science of The Total Environment*, vol. 665, pp. 1103–1110, 2019, ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2019.02.041. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0048969719305212.

[28] J. Johnson. "Selection of materials for uv optics." (Dec. 2008), [Online]. Available: https://wp.optics.arizona.edu/optomech/wp-content/uploads/sites/53/2016/10/Tutorial1_james-Johnson.pdf (visited on 06/13/2023).

[29] A. Schreiber, E. Arnold, B. Kuehn, and F. Schilling, "Radiation resistance of quartz glass for vuv discharge lamps," Jul. 2004. DOI: 10.13140/RG.2.1.1516.3605. [Online]. Available: https://www.researchgate.net/publication/282867046_Radiation_Resistance_of_Quartz_Glass_for_VUV_Discharge_Lamps.

[30] M. Quazi, F. M. A., A. S. M. A. Haseeb, F. Yusof, H. Masjuki, and A. Ahmed, "Laser-based surface modifications of aluminum and its alloys," *Critical Reviews in Solid State and Materials Sciences*, vol. 41, pp. 1–26, Oct. 2015. DOI: 10.1080/10408436.2015.1076716.

[31] M. Belloli, M. Cigarini, G. Milesi, P. Mutti, and E. Berni, "Effectiveness of two uv-c light-emitting diodes (led) systems in inactivating fungal conidia on polyethylene terephthalate," *Innovative Food Science Emerging Technologies*, vol. 79, p. 103 050, 2022, ISSN: 1466-8564. DOI: https://doi.org/10.1016/j.ifset.2022.103050. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1466856422001357.

[32] K. Ágoston, "Studying and measuring system for motor base unbalance," *Procedia Manufacturing*, vol. 46, pp. 391–396, 2020, 13th International Conference Interdisciplinarity in Engineering, INTER-ENG 2019, 3–4 October 2019, Targu Mures, Romania, ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2020.03.057. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978920309355.

[33] R. Nave. "NAND-gate Latch." (), [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/nandlatch.html (visited on 06/12/2023).

[34] G. Wright. "DEFINITION debouncing." (), [Online]. Available: https://www.techtarget.com/whatis/definition/debouncing (visited on 06/15/2023).

Datasheets

[35] C. Hass, *GUISlice*, version 0.17.0, 2020. [Online]. Available: https://github.com/ImpulseAdventure/GUIslice.

[36] P. Conti, *GUISlice*, version 0.17.b24, 2022. [Online]. Available: https://github.com/ImpulseAdventure/GUIslice-Builder.

[37] L. Hennigs, *Button2*, version 2.2.2, 2022. [Online]. Available: https://github.com/LennartHennigs/Button2/.

[38] L. Hennigs, *ESP Rotary*, version 2.1.1, 2023. [Online]. Available: https://github.com/LennartHennigs/ESPRotary.

[39] NXP, *I2C-bus specification and user manual*, UM10204, Rev. 7.0, Oct. 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[40] Texas Instruments, *KeyStone Architecture Serial Peripheral Interface (SPI)*, SPRUGP2A, Mar. 2012. [Online]. Available: https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf?ts=1686410692820&ref_url=https%5C%253A%5C%252F%5C%252Fwww.google.com%5C%252F.

[41] Texas Instruments, *KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART)*, SPRUGP1, Nov. 2010. [Online]. Available: https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf?ts=1686421391467&ref_url=https%5C%253A%5C%252F%5C%252Fwww.google.com%5C%252F.

[42] Arduino, *Wire*, version 2.0.0, 2017. [Online]. Available: https://reference.arduino.cc/reference/en/language/functions/communication/wire/.

[43] P. Carlini, P. Edwards, D. Gregor, *et al.*, *The GNU C++ Library Manual*, version 2017 (gnu++1z), 2017. [Online]. Available: https://gcc.gnu.org/onlinedocs/libstdc++/manual/index.html.

[44] Espressif, *ESP32 Arduino Core*, version 2.0.9, 2023. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/.

[45] Arduino, *Arduino*, 2023. [Online]. Available: https://docs.arduino.cc/.

[46] *ESP32-WROVER-B & ESP-32-WROVER-IB Datasheet*, 5, Rev. A, MORNSUN Guangzhou Science & Technology Co., Ltd., 2022. [Online]. Available: https://www.mornsun-power.com/html/pdf/K7805-1000R3L.html.

[47] *Rj45-8p/8c*, Rev. A, Ckmtw(Shenzhen Cankemeng), 2019. [Online]. Available: https://datasheet.lcsc.com/lcsc/1912111437_Ckmtw-Shenzhen-Cankemeng-R-RJ45R08P-C000_C386757.pdf.

[48] *DC/DC Converter. K78xx-1000R3(L) Series. Wide Input voltage Non-Isolated and regulated single output*, 1, Rev. 7, Espressif Systems, 2021. [Online]. Available: https://nl.mouser.com/datasheet/2/891/esp32_wrover_b_datasheet_en-1384674.pdf.

[49] *AMS1117 1A LOW DROPOUT VOLTAGE REGULATOR*, Advanced Monolithic Systems. [Online]. Available: https://datasheet.lcsc.com/lcsc/2304140030_Advanced-Monolithic-Systems-AMS1117-3-3_C6186.pdf.

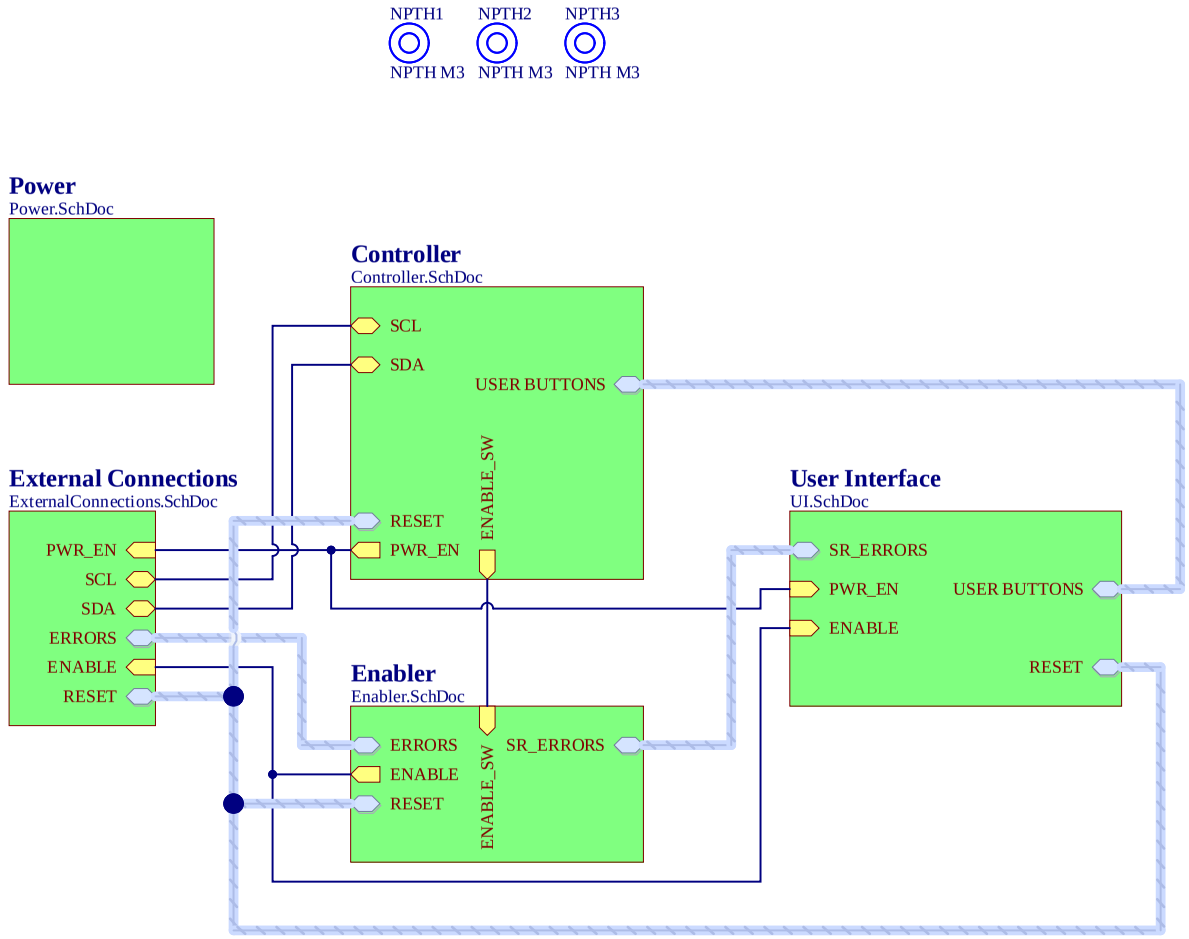
[50] *USBXpress Family CP2102N Datasheet*, 1, Rev. 5, Silicon Labs, 2020. [Online]. Available: https://nl.mouser.com/datasheet/2/368/cp2102n_datasheet-1634912.pdf.

[51] Arduino, *SD*, version 1.2.4, 2019. [Online]. Available: https://github.com/arduino-libraries/SD.

[52] *megaAVR Datasheet*, Microchip Technology Inc., 2020. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf.

A

CAD design of the Control Unit

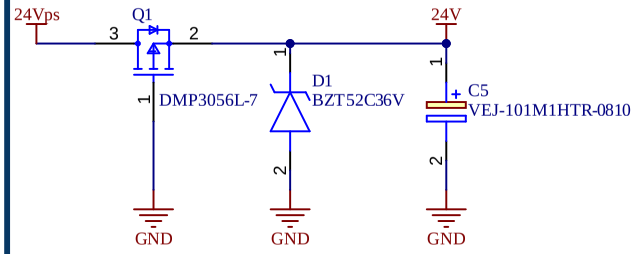


Project:	**UVO - Control Unit.PrjPcb**	Engineer:	
Sheet:	Toplevel	Team Year:	
Date: 16-5-2023	Time: 18:25:16	Sheet 1 of 7	
File:	Toplevel.SchDoc		

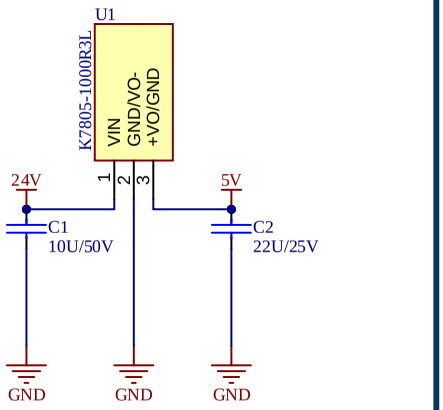
Schieweg 15
2627 AN Delft
The Netherlands
fsteamdelft.nl

FORMULA STUDENT TEAM DELFT

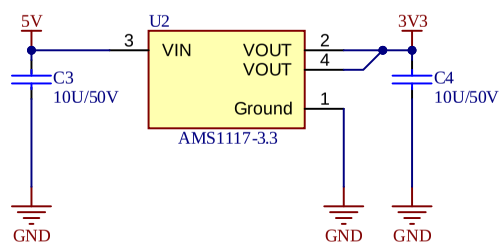
INPUT PROTECTION AND FILTERING



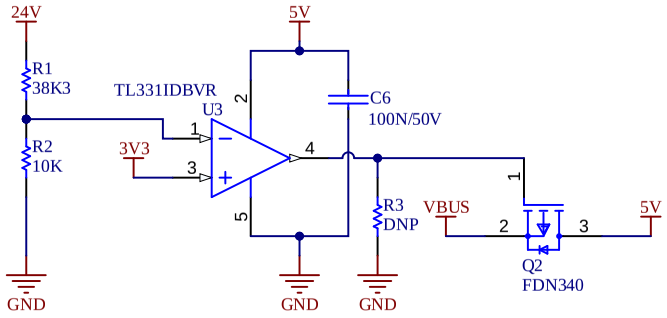
5V GENERATION @ 1000mA



3V3 GENERATION @ 1000mA

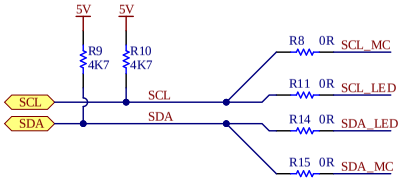


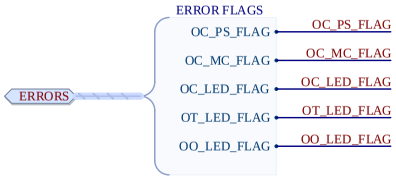
BUS / SUPPLY POWER CHECK

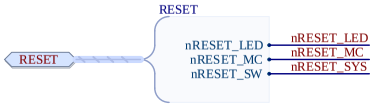


Project: UVO - Control Unit.PrjPcb		Engineer: *	
Sheet:		Team Year:	
Date: 16-5-2023	Time: 18:25:16	Sheet 2 of 7	
File: Power.SchDoc			

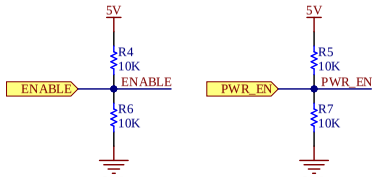
FORMULA STUDENT TEAM DELFT



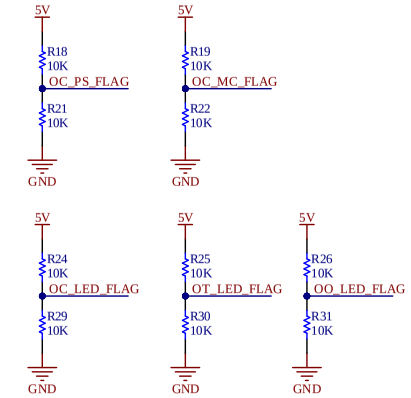




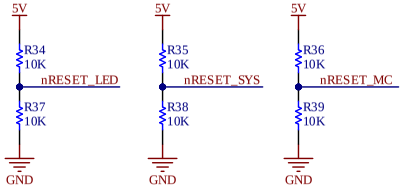
PULL-UP/DOWN FOR ENABLES



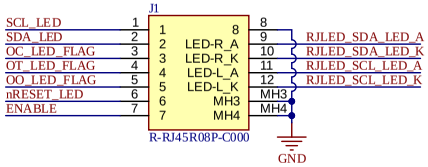
PULL-UP/DOWN FOR FLAGS



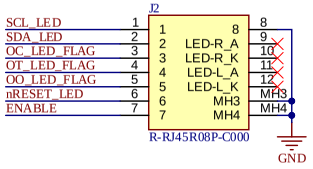
PULL-UP/DOWN FOR RESETS



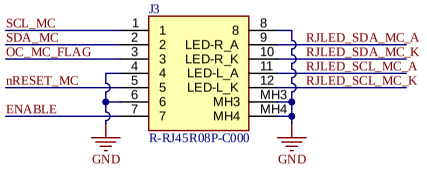
CONNECTIONS TO LED DRIVER TOP



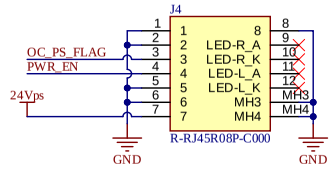
CONNECTIONS TO LED DRIVER BOT



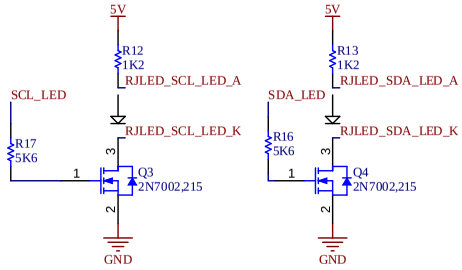
CONNECTIONS TO MOTORCONTROLLER



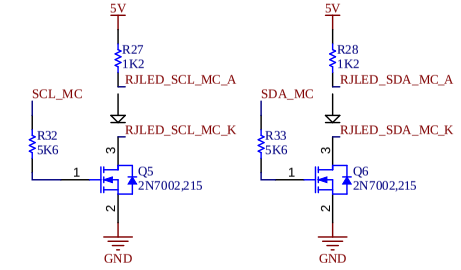
CONNECTIONS TO POWER SUPPLY



RJ45 LEDS FOR LED DRIVER I2C



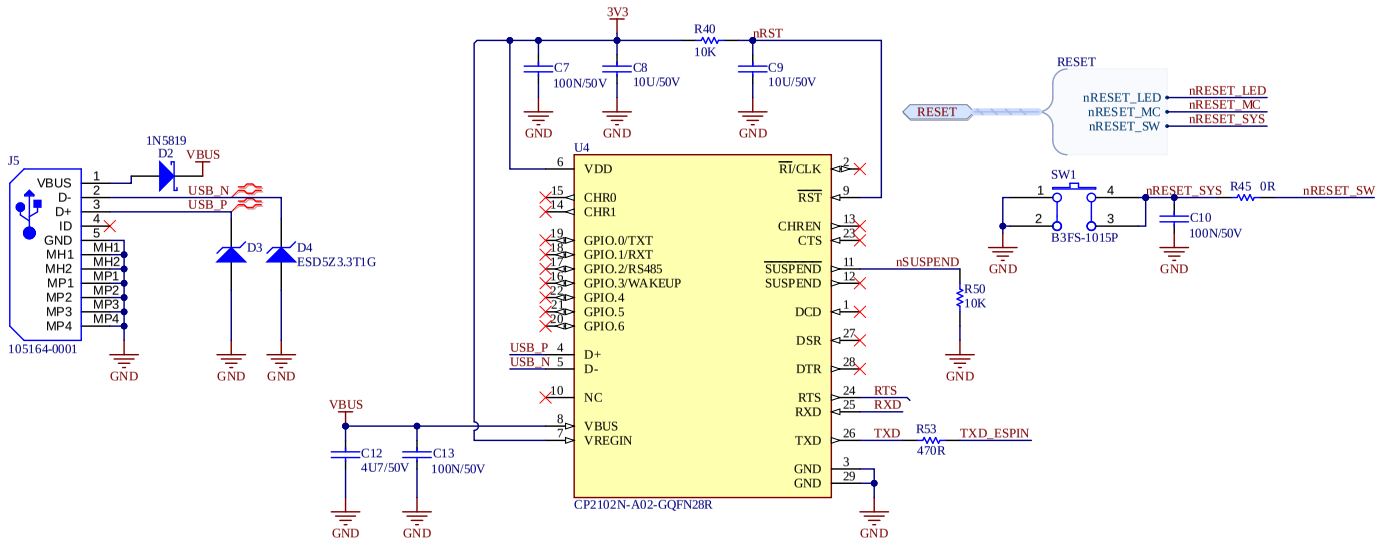
RJ45 LEDS FOR MOTORCONTROLLER I2C



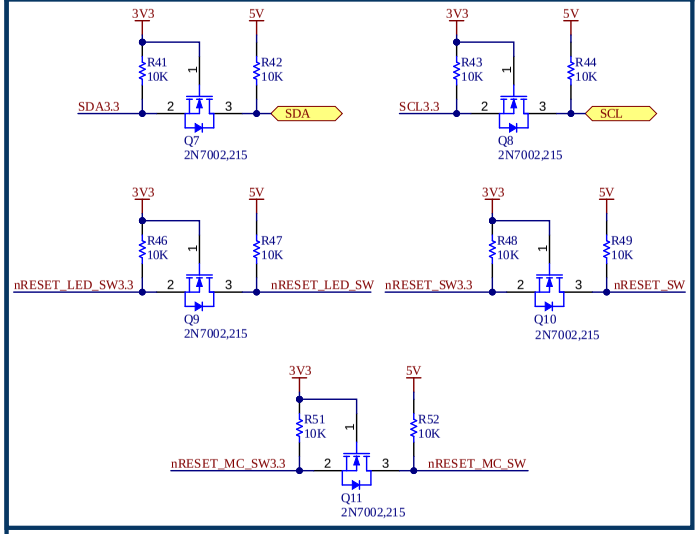
Project: UVO - Control Unit.PrjPcb		Engineer: *	
Sheet:		Team Year:	
Date: 16-5-2023	Time: 18:25:16	Sheet 3 of 7	
File: ExternalConnections.SchDoc			

FORMULA STUDENT TEAM DELFT

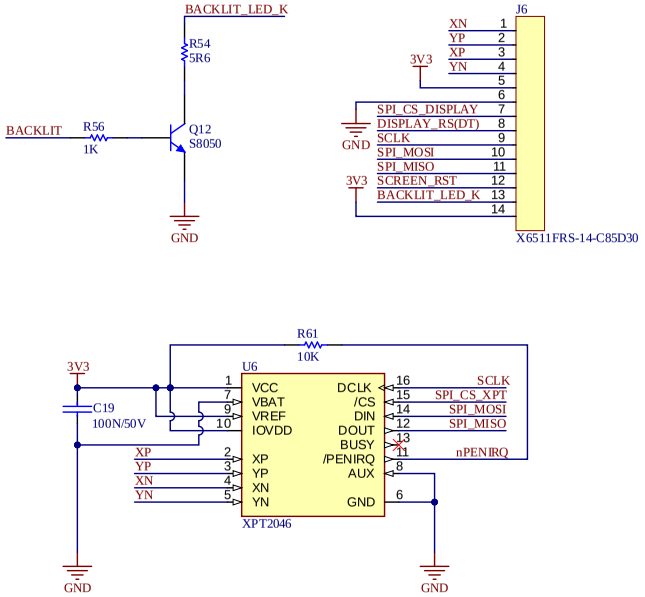
USB ADAPTER CONNECTION AND USB TO TTL INTERFACE



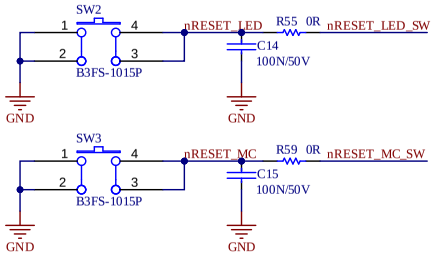
VOLTAGE STEPUP FROM 3V3 TO 5V



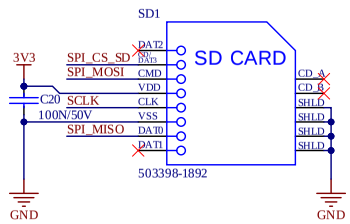
DISPLAY INTERFACE AND TOUCHSCREEN INTERFACE



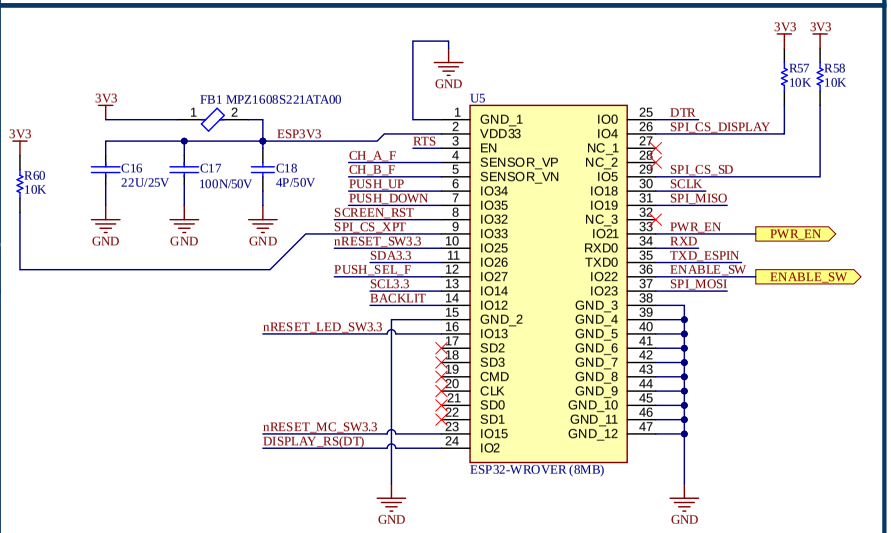
RESET HW AND SW INTERCONNECT

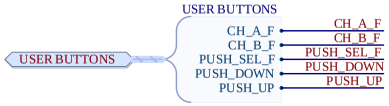


SD MEMORY STORAGE / DATA LOG



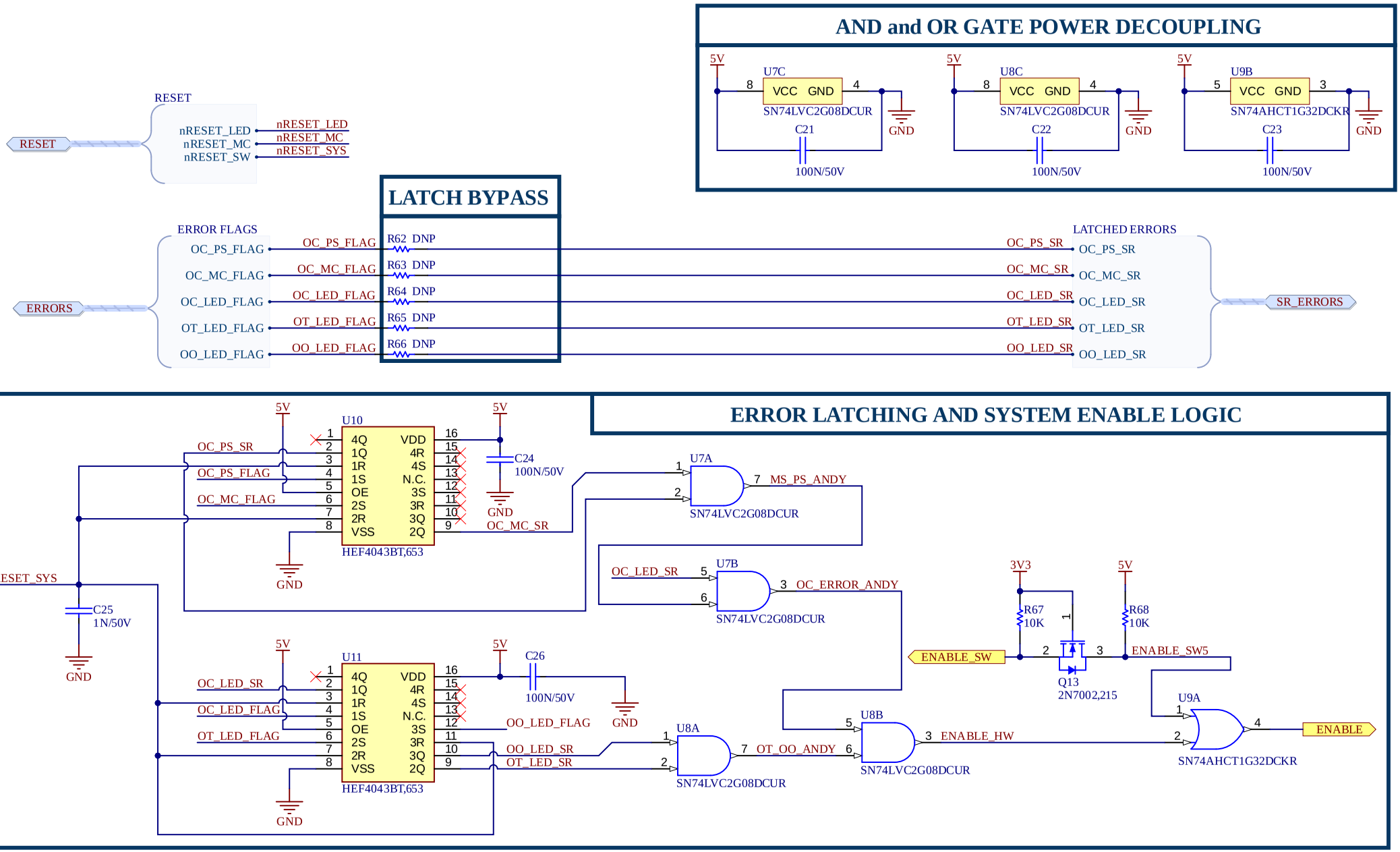
ESP32 - MICROCONTROLLER





Project: UVO - Control Unit.PrjPcb		Engineer: *	
Sheet:		Team Year:	
Date: 16-5-2023	Time: 18:25:17	Sheet 4 of 7	
File: Controller.SchDoc			

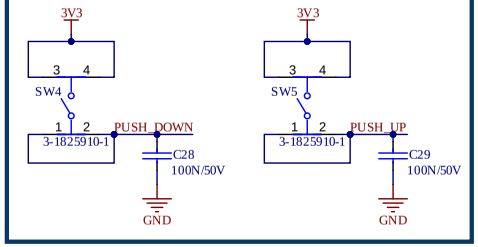
FORMULA STUDENT TEAM DELFT

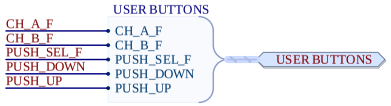


Project:	**UVO - Control Unit.PrjPcb**	Engineer:	*
Sheet:		Team Year:	
Date:	16-5-2023 Time: 18:25:17	Sheet	5 of 7
File:	Enabler.SchDoc		

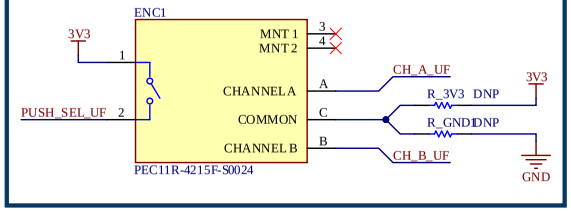
FORMULA STUDENT TEAM DELFT

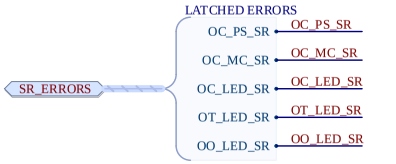
UP AND DOWN MENU SCREEN BUTTONS

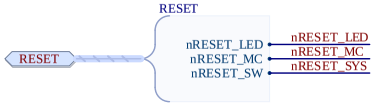




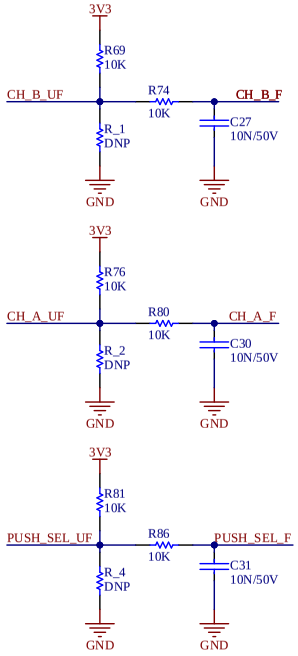
ROTARY PUSH ENCODER FOR MENU



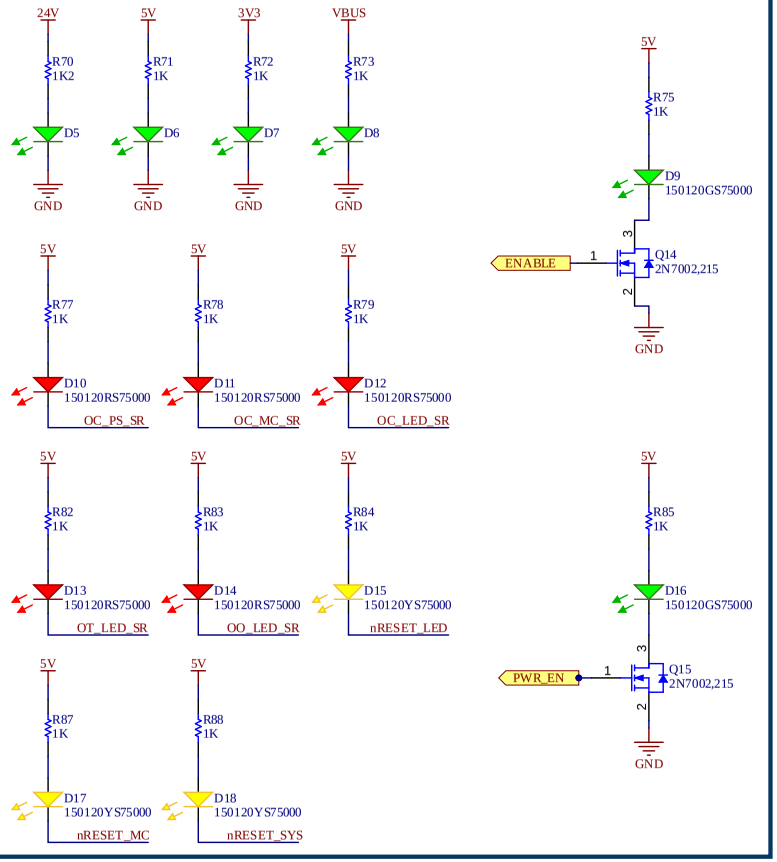




ROTARY ENCODER FILTERING



SYSTEM STATUS LED INDICATORS



Project: UVO - Control Unit.PrjPcb		Engineer: *	
Sheet:		Team Year:	
Date: 16-5-2023	Time: 18:25:17	Sheet 6 of 7	
File: UI.SchDoc			

FORMULA STUDENT TEAM DELFT

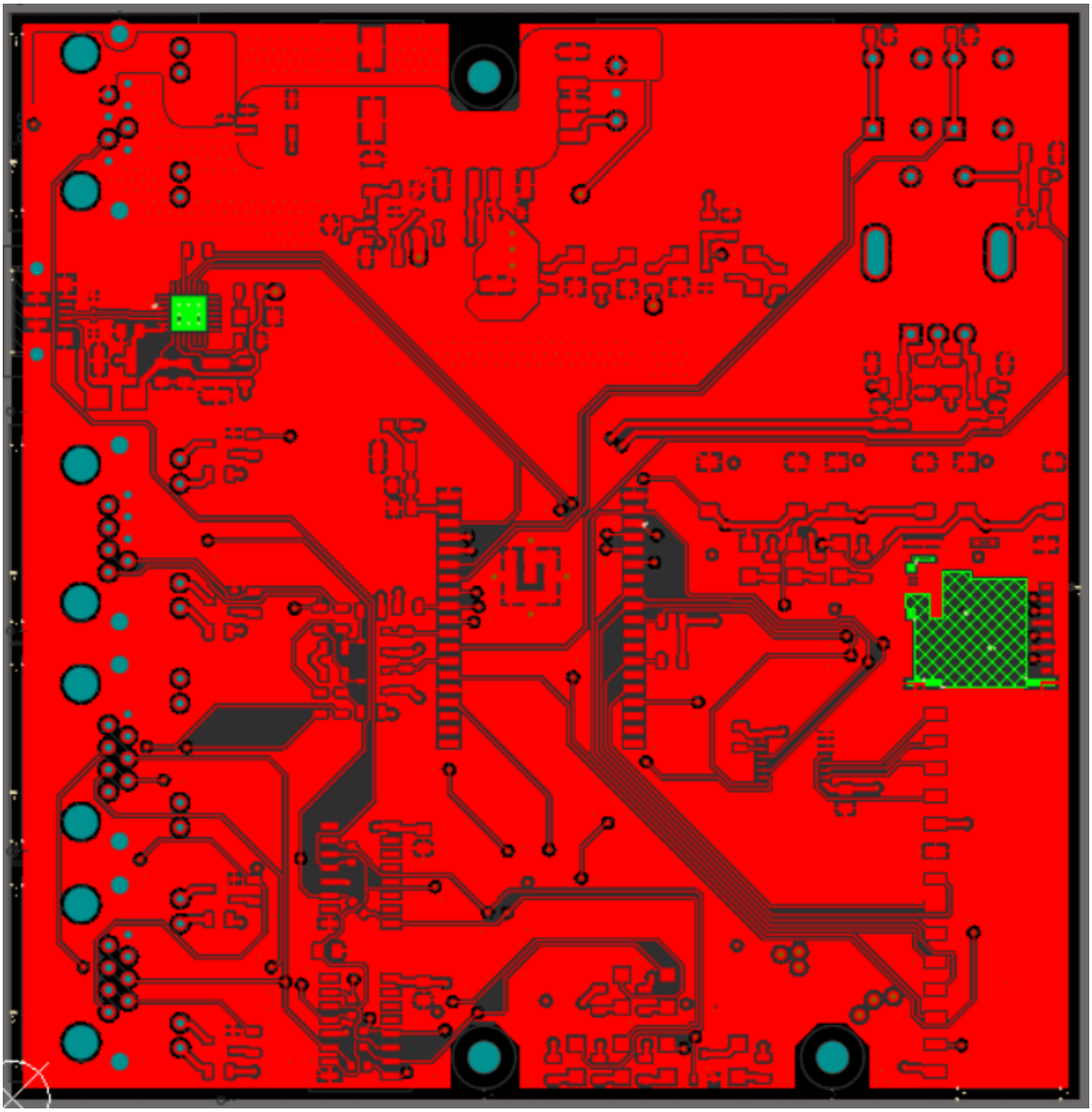


Figure A.1: Top layer of the PCB routing design

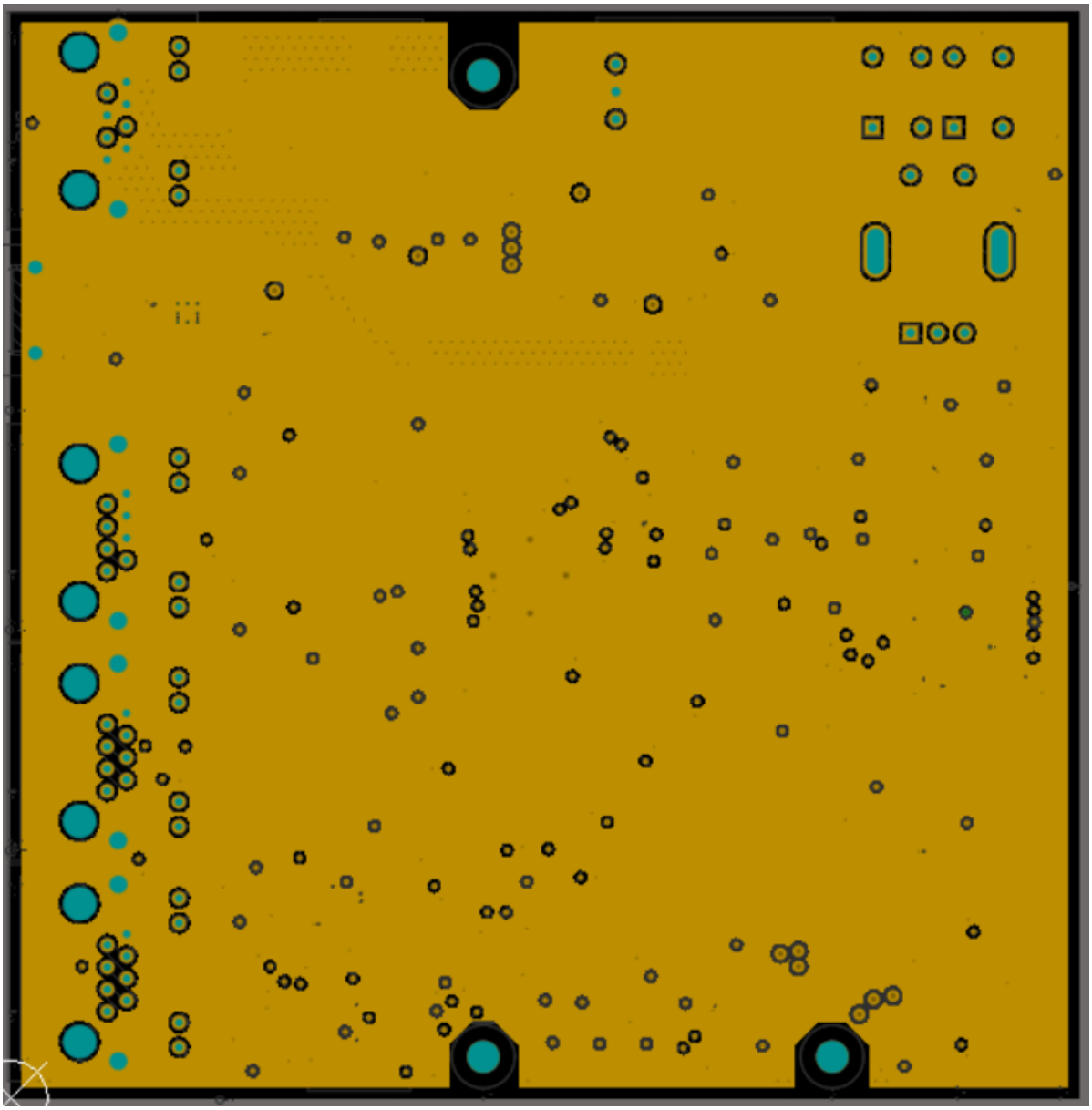


Figure A.2: Ground layer of the PCB routing design

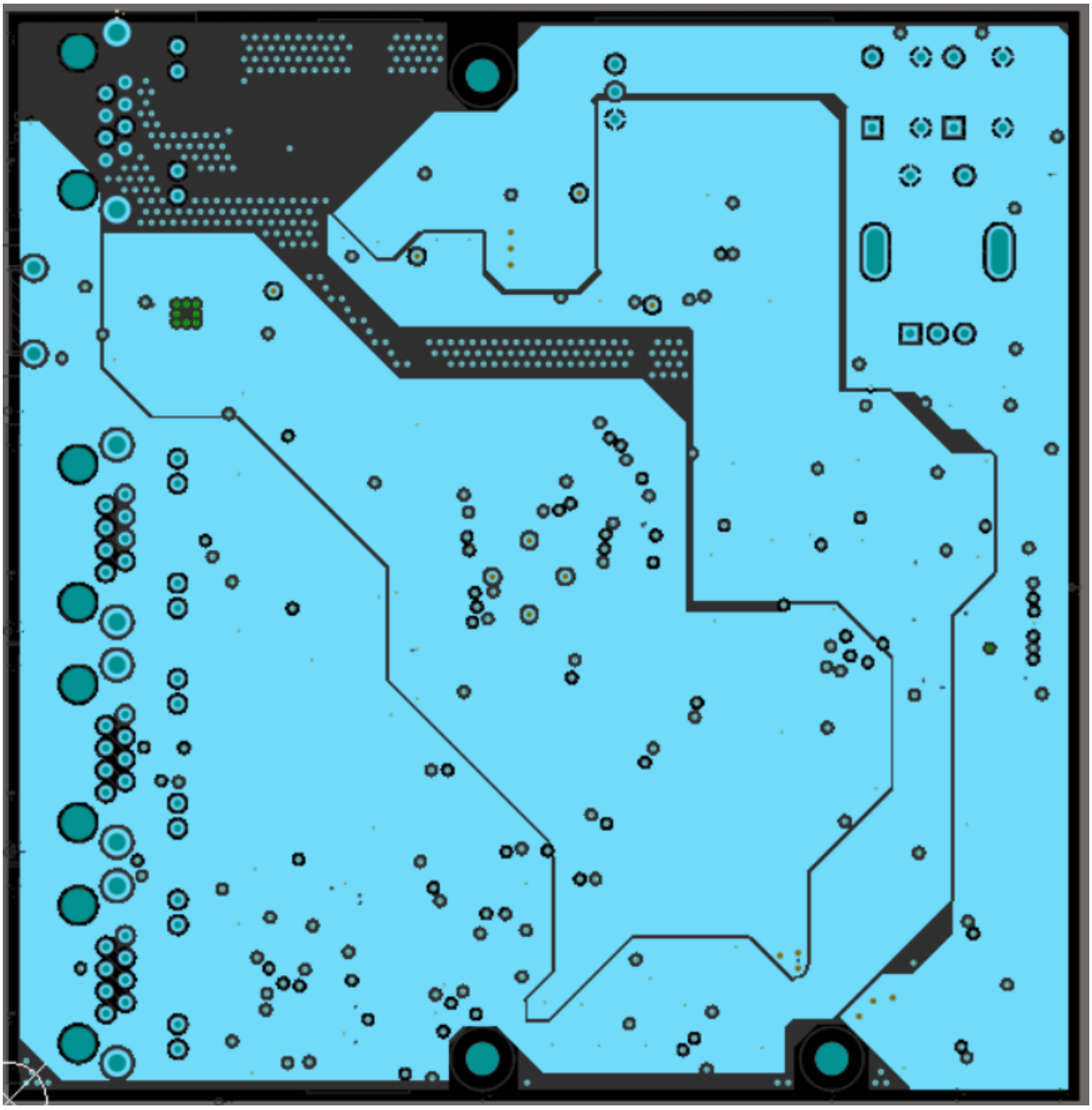


Figure A.3: Power layer of the PCB routing design

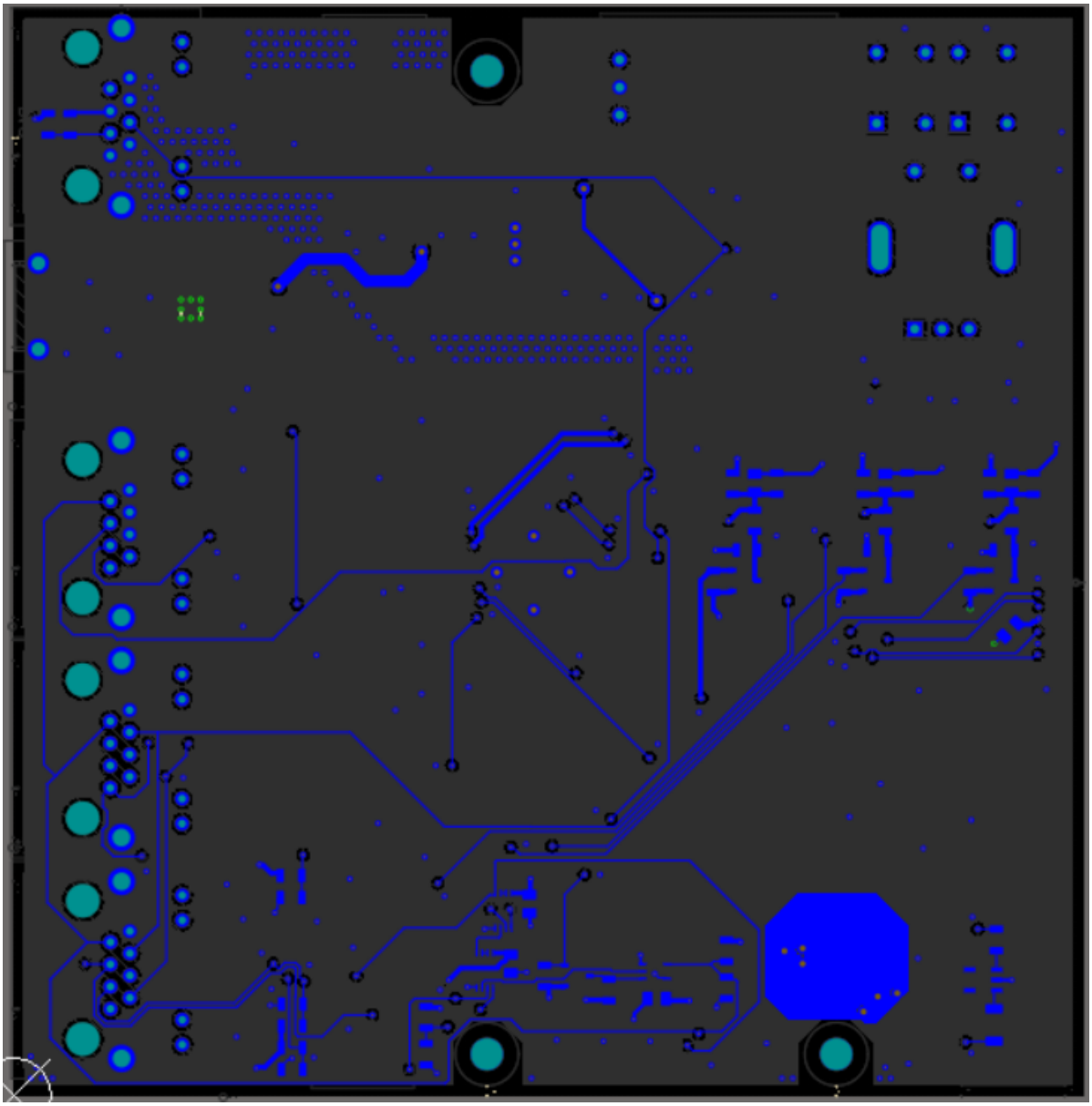


Figure A.4: Bottom layer of the PCB routing design

B

Test Setups

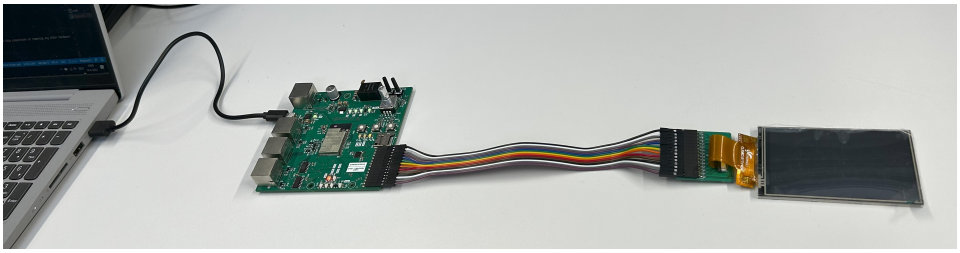


Figure B.1: On the left, the Control Unit is connected to the laptop using USB. On the right, it is connected to the screen. This would be the test setup if uploading to the Control Unit would have worked.

Figure B.2: The UI screen functionality was developed using a test setup created by our colleague Rik Imbens. The laptop is connected via USB to the test setup. The circuit used in the Control Unit is the same as that from the test setup

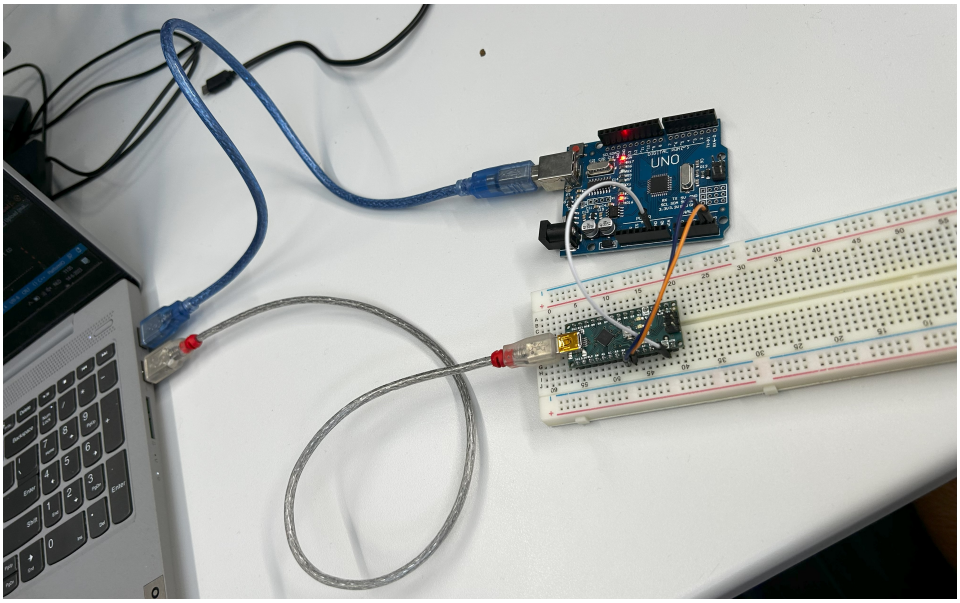


Figure B.3: Two Arduinos (one Uno and one Nano) are connected to eachother at pins A4, A5 and GND. This setup is representative for the final setup, as the arduinos employ the same code, only the compilation differs.

C

Communication Protocol

C.1. General Tokens

Table C.1: Package Type Tokens Definitions

Package Type Tokens	Value
REQUEST_SENSOR_DATA	S
SET_DRIVER_INTENSITY	D
SET_VARIABLE_RESISTOR	R
SEND_ERROR_FLAG	E

Table C.2: General Address and Token Definitions

General Definitions	Decimal Value
Control Unit I^2C Address	40
Top LED Controller I^2C Address	60
Bottom LED Controller I^2C Address	61
Motor Controller I^2C Address	80
Acknowledge Token	2
Not Acknowledge Token	4
Invalid Token	255

C.2. LED Driver Tokens

All the tokens are the same for the bottom and top LED driver. That way, the same software can be used for the top and bottom units, only initialized with different I^2C addresses.

Table C.3: The Sensor Tokens of the LED Driver

Sensor Tokens	Decimal Value
CURRENTSENSOR_255nm	1
CURRENTSENSOR_275nm	2
CURRENTSENSOR_285nm	4
CURRENTSENSOR_395nm	8
SEED_TEMPERATURE_SENSOR	16
ULTRAVIOLET_INTENSITY_SENSOR	32
OZONE_SENSOR	64
LEDS_TEMPERATURE_SENSOR	128

Table C.4: Driver tokens of the LED Driver

Driver Tokens	Decimal Value
PWM_255nm	1
PWM_265nm	2
PWM_275nm	4
PWM_395nm	8

Table C.5: Variable Resistor Tokens of the LED Driver

Variable Resistor Tokens	Decimal Value
CURRENTSENSOR_285nm	1
CURRENTSENSOR_275nm	2
CURRENTSENSOR_255nm	4
CURRENTSENSOR_395nm	8
BOOSTCONVERTER_255nm	3
BOOSTCONVERTER_275nm	6
BOOSTCONVERTER_285nm_395nm	12
ULTRAVIOLETSENSOR	7
OZONSENSOR	14
SEEDTEMPERATURESENSOR	28

C.3. Motor Controller

Table C.6: Sensor Tokens of the Motor Controller

Sensor Tokens	Decimal Value
CURRENTSENSOR_MOTOR	1

Table C.7: Driver Tokens of the Motor Controller

Driver Tokens	Decimal Value
PWM_MOTOR	1

D

MATLAB Code

D.1. Plotting transmission against quartz plate thickness for 260nm

```
% SCRIPT TO GENERATE TRANSMISSION PERCENTAGES FOR A RANGE OF PLATE THICKNESSES FOR A GIVEN WAVELENGTH (260NM)
% Author = M. Mazurovs & E. Ergul
%
%
% Generate L values
L = linspace(0, 10, 100); % Adjust the range and number of points as needed

alfa = -ln(0.8); %Calculate absorption factor given transmission percentage for 260nm for a 1 mm thick quartz plate


% Calculate I/I_0 values
I0 = 1; % Initial intensity
I = I0 * exp(-alfa * L);

% Plot the results
plot(L, I/I0, 'b-', 'LineWidth', 2);
xlabel('L [mm]');
ylabel('T [%]');
title('Transmission plot of 260 nm UVC light through different thickness of quartz material');
grid on;
```

E

C++ code

Using the PlatformIO extension (PIO Core 6.1.7, PIO Home 3.4.4) in Visual Studio Code. C++ compiler and standard gnu++1z.
The entire code as submitted in the thesis can be found in
`https://github.com/Magmoc/UVO_ControlUnit/tree/version-thesis-submission`.

The version of the thesis defense can be found in
`https://github.com/Magmoc/UVO_ControlUnit/tree/Thesis-Defense-code`.

Dependency List of Libraries

```
Dependency Graph
|-- TFT_eSPI @ 2.5.30
|-- GUIslice @ 0.17.0
|-- Chrono @ 1.2.0
|-- ESP Rotary @ 2.1.1
|-- Button2 @ 2.2.2
|-- Wire @ 2.0.0
|-- SPI @ 2.0.0
|-- FS @ 2.0.0
|-- SPIFFS @ 2.0.0
```

Project Structure

```
C:.
|   INSTALL
|   platformio.ini
|   README.md
|   TODO
+---include
|   |   logger.hpp
|   |   main.hpp
|   |
|   +---components
|   |   |   sd_interface.hpp
|   |   |   settings.hpp
|   |   |
|   |   +---communication
|   |   |       communication_protocol.hpp
|   |   |       I2C_interface.hpp
|   |   |       main_controller_communication_interface.hpp
|   |   |       sensor.hpp
|   |   |
|   |   \---GUI
|   |           button.hpp
|   |           color_schemes.hpp
|   |           GUISliceBuilder_GSLC.hpp
```

```
|   |           GUISlice_references_content.hpp
|   |           GUISlice_screen.hpp
|   |           rotary_encoder.hpp
|   |           screen.hpp
|   |           Seven_Segment16pt7b.h
|   |
|   \---modules
|       \---control_module
|               main_controller.hpp
|               main_controller_defines.hpp
|
+---src
|   |   logger.cpp
|   |   main.cpp
|   |
|   +---communication_test
|   |       I2C_communication_test.cpp
|   |       I2C_communication_test.hpp
|   |
|   +---components
|   |   |   sd_interface.cpp
|   |   |   sensor.cpp
|   |   |   settings.cpp
|   |   |
|   |   +---communication
|   |   |       I2C_interface.cpp
|   |   |       main_controller_communication_interface.cpp
|   |   |
|   |   \---GUI
|   |           button.cpp
|   |           GUISlice_screen.cpp
|   |           screen.cpp
|   |
|   \---modules
|           main_controller.cpp
|
\---test
        README
```

import_GUISliceMenu.py

```python
# ***********************************************
#     EE3L11: Bachelor Graduation Project
#   GROUP M: UVC SEED STERILIZATION
#   SUBGROUP: SOFTWARE AND CONTROL
#     MEMBERS: Erman Erg l , Erik van Weelderen
#
#     BY ERIK VAN WEELDEREN
#     DATE: 16-6-2023
# ***********************************************

import os, shutil

DIRNAME = os.path.dirname(__file__)
GUI_SLICE_BUILDER_FILE = None

headers = list()

for file in os.listdir(DIRNAME):
  if file.endswith(".h"):
    if "GSLC" in file:
      GUI_SLICE_BUILDER_FILE = file
    else:
      headers.append(file)
```

```
24
25 if not GUI_SLICE_BUILDER_FILE:
26   raise Exception("Please generate new code within GUISLiceBuilder")
27
28 GUI_SLICE_BUILDER_FILEPATH = os.path.join(DIRNAME, GUI_SLICE_BUILDER_FILE)
29
30 with open(GUI_SLICE_BUILDER_FILEPATH) as f:
31   contents = f.read()
32
33
34 os.remove(GUI_SLICE_BUILDER_FILEPATH)
35 print(f"Removed {GUI_SLICE_BUILDER_FILE}")
36
37 # Replace function
38 FUNC_DEF = "void InitGUIslice_gen()"
39 FUNC_REDEF = "inline " + FUNC_DEF
40
41 contents = contents.replace(FUNC_DEF, FUNC_REDEF)
42
43 # replace the defines in the definitions part
44 DEFINES_START = """// -------------------------------------------------
45 // Create element storage
46 // -------------------------------------------------"""
47
48 DEFINES_END = """// -------------------------------------------------
49 // Program Globals
50 // -------------------------------------------------
51 """
52
53 DEFINES_IDENTIFIER = "gslc_ts"
54 DEFINES_REDEF = "inline " + DEFINES_IDENTIFIER
55
56 defines_start_idx = contents.find(DEFINES_START)
57 defines_end_idx = contents.find(DEFINES_END)
58
59 part_in_which_to_replace = contents[defines_start_idx:defines_end_idx]
60 replaced_parts = part_in_which_to_replace.replace(DEFINES_IDENTIFIER,
     DEFINES_REDEF)
61 contents = contents.replace(part_in_which_to_replace, replaced_parts)
62
63 GSLC_output_filename = GUI_SLICE_BUILDER_FILE.replace(".h", ".hpp")
64
65 GSLC_outpath = os.path.join(DIRNAME, GSLC_output_filename)
66 with open(GSLC_outpath, "w+") as f:
67   f.write(contents)
68
69
70
71
72 ############################################################
73 #
74 #          IMPORT INO FILE
75 #
76 ############################################################
77
78 REFERENCES_START = """// -------------------------------------------------
```

```
// Program Globals
// ------------------------------------------------
"""
REFERENCES_END = "//<Tick Callback !End!>"

for file in os.listdir(DIRNAME):
  if file.endswith(".ino"):
    GUI_SLICE_BUILDER_INO = file


GUI_SLICE_BUILDER_INO_FILEPATH = os.path.join(DIRNAME,
    GUI_SLICE_BUILDER_INO)

with open(GUI_SLICE_BUILDER_INO_FILEPATH) as f:
  contents = f.read()

os.remove(GUI_SLICE_BUILDER_INO_FILEPATH)
print(f"Removed {GUI_SLICE_BUILDER_INO}")



references_start_idx = contents.find(REFERENCES_START)
references_end_idx = contents.find(REFERENCES_END)

references_content = contents[references_start_idx:references_end_idx]

REFERENCES_HEADER_FILENAME = "GUISlice_references_content.hpp"
REFERENCES_HEADER_PATH = os.path.join(DIRNAME, REFERENCES_HEADER_FILENAME)

HEADER_GUARD_START_TEXT = """#ifndef _GUISLICE_REFERENCES_CONTENT_HPP
#define _GUISLICE_REFERENCES_CONTENT_HPP
"""

HEADER_GUARD_END_TEXT = """
#endif"""

with open(REFERENCES_HEADER_PATH, "w+") as f:
  f.write(HEADER_GUARD_START_TEXT)
  f.write(references_content)
  f.write(HEADER_GUARD_END_TEXT)

def find(name, path):
  for root, dirs, files in os.walk(path):
    if name in files:
      return os.path.join(root, name)


INCLUDE_DIR = "include"
INCLUDE_DIR_PATH = os.path.abspath(os.path.join(DIRNAME, os.pardir,
    INCLUDE_DIR))

def replace(filename, search_directory):
  file_to_replace = find(filename, search_directory)
  try:
    outpath = os.path.join(DIRNAME, filename)
    os.replace(outpath, file_to_replace)
```

```
133     print(f"Succesfully replaced {outpath}!")
134   except:
135     print(f"Failed to replace {outpath}!")
136 
137 replace(GSLC_output_filename, INCLUDE_DIR_PATH)
138 
139 replace(REFERENCES_HEADER_FILENAME, INCLUDE_DIR_PATH)
140 
141 for header in headers:
142   replace(header, INCLUDE_DIR_PATH)
143 
144 # TODO Export everything with same _SETUP_ into cpp array with said name
```

Listing E.1: import_GUISliceMenu.py

platformio.ini

```
1  ; PlatformIO Project Configuration File
2  ;
3  ;   Build options: build flags, source filter
4  ;   Upload options: custom upload port, speed and extra flags
5  ;   Library options: dependencies, extra library storages
6  ;   Advanced options: extra scripting
7  ;
8  ; Please visit documentation for the other options and examples
9  ; https://docs.platformio.org/page/projectconf.html
10 
11 [env]
12 build_flags =
13   -std=gnu++1z
14   -Os
15 
16   -ffunction-sections
17   -fdata-sections
18   -Wl,--gc-sections
19 upload_speed = 921600
20 lib_ldf_mode = deep+
21 build_unflags = -std=gnu++11
22 
23 [env:main-controller]
24 framework = arduino
25 platform = espressif32
26 board = esp-wrover-kit
27 monitor_speed = 9600
28 lib_deps =
29   bodmer/TFT_eSPI@^2.5.30
30   impulseadventure/GUIslice@^0.17.0
31   thomasfredericks/Chrono@^1.2.0
32   lennarthennigs/ESP Rotary@^2.1.1
33   lennarthennigs/Button2@^2.2.2
34 build_unflags = ${env.build_unflags}
35 build_flags =
36   ${env.build_flags}
37   -D MAIN_CONTROLLER_MODULE
38 
39   ; Options: USE_SCREEN, USE_BUTTONS, USE_COMMUNICATION_INTERFACE
40   -D USE_SCREEN
```

```
  -D DEBUG_MODE

  -D BOARD_HAS_PSRAM
  -mfix-esp32-psram-cache-issue

  -D USER_SETUP_LOADED=1

  -D ST7796_DRIVER=1

  -D TFT_RST=32
  -D TFT_MISO=19
  -D TFT_MOSI=23
  -D TFT_SCLK=18
  -D TFT_CS=4
  -D TOUCH_CS=33


  -D TFT_DC=2
  -D LOAD_GLCD=1
  -D LOAD_FONT2=1
  -D LOAD_FONT4=1
  -D LOAD_FONT6=1
  -D LOAD_FONT7=1
  -D LOAD_FONT8=1
  -D LOAD_GFXFF=1
  -D SMOOTH_FONT=1
  -D SPI_FREQUENCY=27000000

[env:arduino]
framework = arduino
platform = atmelavr
board = uno
lib_ldf_mode = chain+
lib_ignore = bodmer/TFT_eSPI@^2.5.30
build_unflags = ${env.build_unflags}
build_flags =
  ${env.build_flags}

  -D DEBUG_MODE
  -D SLAVE_TEST
monitor_speed = 9600
lib_deps =
  impulseadventure/GUIslice@^0.17.0
  adafruit/Adafruit GFX Library@^1.11.5
  thomasfredericks/Chrono@^1.2.0
  lennarthennigs/ESP Rotary@^2.1.1
  lennarthennigs/Button2@^2.2.2
```

Listing E.2: platformio.ini

include

logger.hpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
```

```
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#ifndef _LOGGER_HPP
#define _LOGGER_HPP

#endif
```

Listing E.3: include/logger.hpp

main.hpp

```
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#ifndef _MAIN_HPP
#define _MAIN_HPP

#include <Arduino.h>

void setup();
void loop();

#if defined(MAIN_CONTROLLER_MODULE)
  #include "modules/control_module/main_controller.hpp"
  UVO_MainController::MainController Controller;
#else
  #error "Define a module: MAIN_CONTROLLER_MODULE"
#endif

#endif
```

Listing E.4: include/main.hpp

include/components

sd_interface.hpp

```
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#ifndef _SD_INTERFACE_HPP
```

```
#define _SD_INTERFACE_HPP

#endif
```

Listing E.5: include/components/sd_interface.hpp

settings.hpp

```
// ********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ********************************************

#ifndef _SETTINGS_HPP
#define _SETTINGS_HPP

#include <stdint.h>
#include <limits.h>
#include <time.h>

namespace UVO_Components {

  struct s_setupSettings
  {
    bool isUpdated = false;

    uint8_t LED_intensity_255nm = 0;
    uint8_t LED_intensity_275nm = 0;
    uint8_t LED_intensity_285nm = 0;
    uint8_t LED_intensity_395nm = 0;

    uint8_t motor_intensity = 0;

    time_t targetExposureTime = 0;

    int globalSampleFrequencyHz = 10;

    void addSeconds(int seconds){
      // Must be ulong type.
      ulong new_time = targetExposureTime + seconds;

      //clamp
      time_t maxTime = ULONG_MAX;
      time_t minTime = 0;
      new_time = (new_time > maxTime) ? maxTime : new_time;
      new_time = (new_time < minTime) ? minTime : new_time;

      targetExposureTime = new_time;
    }

    void addMinutes(int minutes){
      addSeconds(60*minutes);
    }
```

```
51
52      void addHours(int hours){
53        addMinutes(60*hours);
54      }
55    };
56
57    struct s_systemState {
58      s_setupSettings SetupSettings;
59      volatile bool isUpdated = false;
60      long int elapsedExposureTime = 0;
61
62    };
63
64  }
65
66  #endif
```

Listing E.6: include/components/settings.hpp

include/components/communication

communication_protocol.hpp

```
1  // ********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ********************************************
10
11 #ifndef _COMMUNICATION_PROTOCOL_HPP
12 #define _COMMUNICATION_PROTOCOL_HPP
13
14
15 namespace UVO_CommunicationProtocol {
16   // https://stackoverflow.com/questions/112433/should-i-use-define-enum-
       or-const
17
18   // Sent from Control Unit
19   // [REQUEST_SENSOR_DATA] [SENSOR_TOKEN] REQUEST [value (double) (4 bytes
       )]
20   // [SET_DRIVER_INTENSITY] [DRIVER_TOKEN] [uint_8 intensity (0-255)]
       REQUEST [ack (byte)]
21   // [SET_VARIABLE_RESISTOR] [RESISTOR_TOKEN] [uint_8 intensity (0-255)]
       REQUEST [ack (byte)]
22
23   // Sent to Control Unit
24   // [SEND_ERROR_FLAG] [YOUR OWN I2C ADDRESS] [SENSOR_TOKEN] REQUEST [ack
       (byte)]
25
26   const int MAIN_CONTROLLER_ADDRESS = 40;
27   const int TOP_LED_CONTROLLER_ADDRESS = 60;
28   const int BOTTOM_LED_CONTROLLER_ADDRESS = 61;
29   const int MOTOR_CONTROLLER_ADDRESS = 80;
30
```

```
31  typedef unsigned char TToken;
32  typedef TToken TPackageTypeToken;
33  typedef TToken TSensorToken;
34  typedef TToken TDriverToken;
35  typedef TToken TVariableResistorToken;
36
37  const TToken ACK = 2;
38  const TToken NACK = 4;
39
40  const TToken INVALID = 255;
41
42  namespace PackageTypeToken {
43    const TPackageTypeToken REQUEST_SENSOR_DATA = 'S';
44    const TPackageTypeToken SET_DRIVER_INTENSITY = 'D';
45    const TPackageTypeToken SET_VARIABLE_RESISTOR = 'R';
46    const TPackageTypeToken SEND_ERROR_FLAG = 'E';
47  }
48
49  namespace LEDDriverToken {
50    //TODO What do we measure at every sensor? What are the conversions
51
52    namespace SensorToken {
53      const TSensorToken CURRENTSENSOR_255nm = 1;
54      const TSensorToken CURRENTSENSOR_275nm = 2;
55      const TSensorToken CURRENTSENSOR_285nm = 4;
56      const TSensorToken CURRENTSENSOR_395nm = 8;
57
58      const TSensorToken SEED_TEMPERATURE_SENSOR = 16;
59      const TSensorToken ULTRAVIOLET_INTENSITY_SENSOR = 32;
60      // TODO: RENAME TO OZONE
61      const TSensorToken OZON_SENSOR = 64;
62      const TSensorToken LEDS_TEMPERATURE_SENSOR = 128;
63    }
64
65    namespace DriverToken {
66        const TDriverToken PWM_255nm = 1;
67        const TDriverToken PWM_265nm = 2;
68        const TDriverToken PWM_275nm = 4;
69        const TDriverToken PWM_395nm = 8;
70    }
71
72    namespace VariableResistorToken {
73        const TVariableResistorToken CURRENTSENSOR_285nm = 1; // CS1 (
      Current Sensor X)
74        const TVariableResistorToken CURRENTSENSOR_275nm = 2; // CS2
75        const TVariableResistorToken CURRENTSENSOR_255nm = 4; // CS3
76        const TVariableResistorToken CURRENTSENSOR_395nm = 8; // CS4
77
78        const TVariableResistorToken BOOSTCONVERTER_255nm = 3; // BC1 (
      Boost Converter X)
79        const TVariableResistorToken BOOSTCONVERTER_275nm = 6; // BC2
80        const TVariableResistorToken BOOSTCONVERTER_285nm_395nm = 12; //
      BC3
81
82        const TVariableResistorToken ULTRAVIOLETSENSOR = 7; // UVS
83        const TVariableResistorToken OZONSENSOR = 14; // OS (Ozone Sensor)
```

```
84          const TVariableResistorToken SEEDTEMPERATURESENSOR = 28; // TS (
      Temperature Sensor)
85      }
86    }
87
88    namespace MotorControlToken {
89      //TODO What do we measure at every sensor? What are the conversions
90
91      namespace SensorToken {
92        const TSensorToken CURRENTSENSOR_MOTOR = 1;
93      }
94
95      namespace DriverToken {
96        const TDriverToken PWM_MOTOR = 1;
97      }
98    }
99
100
101
102 }
103
104 #endif
```

Listing E.7: include/components/communication/communication_protocol.hpp

I2C_interface.hpp

```
1  // ********************************************
2  //     EE3L11: Bachelor Graduation Project
3  //     GROUP M: UVC SEED STERILIZATION
4  //     SUBGROUP: SOFTWARE AND CONTROL
5  //     MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //     BY ERIK VAN WEELDEREN
8  //     DATE: 16-6-2023
9  // ********************************************
10
11 #ifndef _I2C_INTERFACE_HPP
12 #define _I2C_INTERFACE_HPP
13
14 #include <Wire.h>
15 #include <Arduino.h>
16 #include <SPI.h>
17
18 //TODO MOVE TO SOMEWHERE ELSE
19 #define MAX_MESSAGE_LENGTH 10
20
21 namespace UVO_Components {
22
23 //TODO make I2C interface subclass of TwoWire
24 class I2CInterface
25 {
26 private:
27   int m_SDA_pin;
28   int m_SCL_pin;
29   int m_I2C_address;
30 public:
31   I2CInterface(int t_I2C_address);
```

```
32   // I2CInterface(int t_I2C_address, int t_SDA_pin, int t_SCL_pin);
33   ~I2CInterface();
34
35   void sendMessages(int address, byte* message, int message_length);
36   int requestAndReadAnswer(int I2C_address, byte* receive_message, int
       bytes_requested);
37
38   void onRequest( void (*t_function)(void) );
39   void onReceive( void (*t_function)(int) );
40 };
41
42 }
43
44 #endif
```

Listing E.8: include/components/communication/I2C_interface.hpp

main_controller_communication_interface.hpp

```
1  // ***********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ***********************************************
10
11 #ifndef _MAIN_CONTROLLER_COMMUNICATION_INTERFACE_H
12 #define _MAIN_CONTROLLER_COMMUNICATION_INTERFACE_H
13
14 #include <Wire.h>
15 #include <Arduino.h>
16 #include "components/communication/communication_protocol.hpp"
17 #include "components/communication/I2C_interface.hpp"
18 #include "components/communication/sensor.hpp"
19
20 namespace UVO_MainController {
21
22 class MainCommunicationInterface{
23 private:
24   UVO_Components::I2CInterface m_I2C_Interface;
25   int sendMessageAndReadResponse(int t_I2C_slave_address, byte* t_message,
       int t_message_length, int t_bytes_requested, byte* t_response_data);
26
27 public:
28   MainCommunicationInterface(int t_I2C_address);
29   ~MainCommunicationInterface();
30
31   // TODO IMPLEMENT IF NEEDED
32   // void MainCommunicationInterface::setSDA(int SDA_pin);
33   // void MainCommunicationInterface::setSCL(int SCL_pin);
34
35   void init(void);
36   void update(void);
37   double requestSensorData(UVO_CommunicationProtocol::Sensor t_sensor);
38
```

```
};

}

#endif
```

Listing E.9: include/components/communication/main_controller_communication_interface.hpp

sensor.hpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#ifndef _SENSOR_HPP
#define _SENSOR_HPP

#include "components/communication/communication_protocol.hpp"

namespace UVO_CommunicationProtocol {

struct Sensor {
  int module_address_I2C;
  UVO_CommunicationProtocol::TSensorToken sensorToken;
};

// https://stackoverflow.com/questions/14425262/why-include-guards-do-not-prevent-multiple-function-definitions/14425299#14425299
// Therefore using the inline keyword
namespace sensors {
  namespace TOP_LEDDriver {
    inline const Sensor current_sensor_255nm = {
      .module_address_I2C = UVO_CommunicationProtocol::TOP_LED_CONTROLLER_ADDRESS,
      .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::SensorToken::CURRENTSENSOR_255nm,
    };

    inline const Sensor current_sensor_275nm = {
      .module_address_I2C = UVO_CommunicationProtocol::TOP_LED_CONTROLLER_ADDRESS,
      .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::SensorToken::CURRENTSENSOR_275nm,
    };

    inline const Sensor current_sensor_285nm = {
      .module_address_I2C = UVO_CommunicationProtocol::TOP_LED_CONTROLLER_ADDRESS,
      .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::SensorToken::CURRENTSENSOR_285nm,
    };

```

```
42      inline const Sensor current_sensor_395nm = {
43          .module_address_I2C = UVO_CommunicationProtocol::
        TOP_LED_CONTROLLER_ADDRESS,
44          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::CURRENTSENSOR_395nm,
45      };
46
47      inline const Sensor seed_temperature_sensor = {
48          .module_address_I2C = UVO_CommunicationProtocol::
        TOP_LED_CONTROLLER_ADDRESS,
49          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::SEED_TEMPERATURE_SENSOR,
50      };
51
52      inline const Sensor ultraviolet_intensity_sensor = {
53          .module_address_I2C = UVO_CommunicationProtocol::
        TOP_LED_CONTROLLER_ADDRESS,
54          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::ULTRAVIOLET_INTENSITY_SENSOR,
55      };
56
57      inline const Sensor ozon_sensor = {
58          .module_address_I2C = UVO_CommunicationProtocol::
        TOP_LED_CONTROLLER_ADDRESS,
59          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::OZON_SENSOR,
60      };
61
62      inline const Sensor LEDs_temperature_sensor = {
63          .module_address_I2C = UVO_CommunicationProtocol::
        TOP_LED_CONTROLLER_ADDRESS,
64          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::LEDS_TEMPERATURE_SENSOR,
65      };
66  }
67
68  namespace BOTTOM_LEDDriver {
69      inline const Sensor current_sensor_255nm = {
70          .module_address_I2C = UVO_CommunicationProtocol::
        BOTTOM_LED_CONTROLLER_ADDRESS,
71          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::CURRENTSENSOR_255nm,
72      };
73
74      inline const Sensor current_sensor_275nm = {
75          .module_address_I2C = UVO_CommunicationProtocol::
        BOTTOM_LED_CONTROLLER_ADDRESS,
76          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::CURRENTSENSOR_275nm,
77      };
78
79      inline const Sensor current_sensor_285nm = {
80          .module_address_I2C = UVO_CommunicationProtocol::
        BOTTOM_LED_CONTROLLER_ADDRESS,
81          .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
        SensorToken::CURRENTSENSOR_285nm,
```

```
82      };
83
84      inline const Sensor current_sensor_395nm = {
85        .module_address_I2C = UVO_CommunicationProtocol::
      BOTTOM_LED_CONTROLLER_ADDRESS,
86        .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
      SensorToken::CURRENTSENSOR_395nm,
87      };
88
89      inline const Sensor seed_temperature_sensor = {
90        .module_address_I2C = UVO_CommunicationProtocol::
      BOTTOM_LED_CONTROLLER_ADDRESS,
91        .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
      SensorToken::SEED_TEMPERATURE_SENSOR,
92      };
93
94      inline const Sensor ultraviolet_intensity_sensor = {
95        .module_address_I2C = UVO_CommunicationProtocol::
      BOTTOM_LED_CONTROLLER_ADDRESS,
96        .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
      SensorToken::ULTRAVIOLET_INTENSITY_SENSOR,
97      };
98
99      inline const Sensor ozon_sensor = {
100       .module_address_I2C = UVO_CommunicationProtocol::
      BOTTOM_LED_CONTROLLER_ADDRESS,
101       .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
      SensorToken::OZON_SENSOR,
102     };
103
104     inline const Sensor LEDs_temperature_sensor = {
105       .module_address_I2C = UVO_CommunicationProtocol::
      BOTTOM_LED_CONTROLLER_ADDRESS,
106       .sensorToken = UVO_CommunicationProtocol::LEDDriverToken::
      SensorToken::LEDS_TEMPERATURE_SENSOR,
107     };
108
109   }
110
111   namespace motor_controller{
112     inline const Sensor current_sensor_motor = {
113       .module_address_I2C = UVO_CommunicationProtocol::
      MOTOR_CONTROLLER_ADDRESS,
114       .sensorToken = UVO_CommunicationProtocol::MotorControlToken::
      SensorToken::CURRENTSENSOR_MOTOR,
115     };
116   }
117 };
118
119 }
120
121 #endif
```

Listing E.10: include/components/communication/sensor.hpp

include/components/GUI

button.hpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#ifndef _BUTTON_HPP
#define _BUTTON_HPP

#include <Button2.h>

#endif
```

Listing E.11: include/components/GUI/button.hpp

color_schemes.hpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************


#ifndef _COLOR_SCHEMES_HPP
#define _COLOR_SCHEMES_HPP

//TODO Can be used to implement color schemes for the code.
// such as determining what the selected colors should be.

#endif
```

Listing E.12: include/components/GUI/color_schemes.hpp

GUISlice_references_content.hpp

```
#ifndef _GUISLICE_REFERENCES_CONTENT_HPP
#define _GUISLICE_REFERENCES_CONTENT_HPP
// ------------------------------------------------
// Program Globals
// ------------------------------------------------

// Save some element references for direct access
//<Save_References !Start!>
gslc_tsElemRef* m_pElem_ERROR_Error1= NULL;
gslc_tsElemRef* m_pElem_ERROR_Error2= NULL;
gslc_tsElemRef* m_pElem_ERROR_Error3= NULL;
gslc_tsElemRef* m_pElem_ERROR_Error4= NULL;
```

```
13 gslc_tsElemRef* m_pElem_MONITOR_Current_255nm= NULL;
14 gslc_tsElemRef* m_pElem_MONITOR_Current_275nm= NULL;
15 gslc_tsElemRef* m_pElem_MONITOR_Current_285nm= NULL;
16 gslc_tsElemRef* m_pElem_MONITOR_Current_395nm= NULL;
17 gslc_tsElemRef* m_pElem_MONITOR_Dosis= NULL;
18 gslc_tsElemRef* m_pElem_MONITOR_Hours= NULL;
19 gslc_tsElemRef* m_pElem_MONITOR_Intensity= NULL;
20 gslc_tsElemRef* m_pElem_MONITOR_Intensity127= NULL;
21 gslc_tsElemRef* m_pElem_MONITOR_Minutes= NULL;
22 gslc_tsElemRef* m_pElem_MONITOR_Pause= NULL;
23 gslc_tsElemRef* m_pElem_MONITOR_Seconds= NULL;
24 gslc_tsElemRef* m_pElem_MONITOR_Stop= NULL;
25 gslc_tsElemRef* m_pElem_MONITOR_Stop136= NULL;
26 gslc_tsElemRef* m_pElem_MONITOR_Temperature_LED_Bottom= NULL;
27 gslc_tsElemRef* m_pElem_MONITOR_Temperature_LED_Top= NULL;
28 gslc_tsElemRef* m_pElem_MONITOR_Temperature_Seed= NULL;
29 gslc_tsElemRef* m_pElem_SETUP_Dosis_255nm= NULL;
30 gslc_tsElemRef* m_pElem_SETUP_Dosis_275nm= NULL;
31 gslc_tsElemRef* m_pElem_SETUP_Dosis_285nm= NULL;
32 gslc_tsElemRef* m_pElem_SETUP_Dosis_395nm= NULL;
33 gslc_tsElemRef* m_pElem_SETUP_Hours= NULL;
34 gslc_tsElemRef* m_pElem_SETUP_Intensity_255nm= NULL;
35 gslc_tsElemRef* m_pElem_SETUP_Intensity_275nm= NULL;
36 gslc_tsElemRef* m_pElem_SETUP_Intensity_285nm= NULL;
37 gslc_tsElemRef* m_pElem_SETUP_Intensity_395nm= NULL;
38 gslc_tsElemRef* m_pElem_SETUP_Minutes= NULL;
39 gslc_tsElemRef* m_pElem_SETUP_MotorIntensity= NULL;
40 gslc_tsElemRef* m_pElem_SETUP_Seconds= NULL;
41 gslc_tsElemRef* m_pElem_SETUP_Start= NULL;
42 //<Save_References !End!>
43
44 // Define debug message function
45 static int16_t DebugOut(char ch) { if (ch == (char)'\n') Serial.println(""
       ); else Serial.write(ch); return 0; }
46
47 // ------------------------------------------------
48 // Callback Methods
49 // ------------------------------------------------
50 //<Button Callback !Start!>
51 //<Button Callback !End!>
52 //<Checkbox Callback !Start!>
53 //<Checkbox Callback !End!>
54 //<Keypad Callback !Start!>
55 //<Keypad Callback !End!>
56 //<Spinner Callback !Start!>
57 //<Spinner Callback !End!>
58 //<Listbox Callback !Start!>
59 //<Listbox Callback !End!>
60
61 // Scanner drawing callback function
62 // - This is called when E_ELEM_SCAN is being rendered
63 bool CbDrawScanner(void* pvGui,void* pvElemRef,gslc_teRedrawType eRedraw)
64 {
65   int nInd;
66
67   // Typecast the parameters to match the GUI and element types
```

```
68    gslc_tsGui*     pGui     = (gslc_tsGui*)(pvGui);
69    gslc_tsElemRef* pElemRef = (gslc_tsElemRef*)(pvElemRef);
70    gslc_tsElem*    pElem    = gslc_GetElemFromRef(pGui,pElemRef);
71
72    // Create shorthand variables for the origin
73    int16_t  nX = pElem->rElem.x;
74    int16_t  nY = pElem->rElem.y;
75
76    // Draw the background
77    gslc_tsRect rInside = pElem->rElem;
78    rInside = gslc_ExpandRect(rInside,-1,-1);
79    gslc_DrawFillRect(pGui,rInside,pElem->colElemFill);
80
81    // Enable localized clipping
82    gslc_SetClipRect(pGui,&rInside);
83
84    //TODO - Add your drawing graphic primitives
85
86    // Disable clipping region
87    gslc_SetClipRect(pGui,NULL);
88
89    // Draw the frame
90    gslc_DrawFrameRect(pGui,pElem->rElem,pElem->colElemFrame);
91
92    // Clear the redraw flag
93    gslc_ElemSetRedraw(&m_gui,pElemRef,GSLC_REDRAW_NONE);
94
95    return true;
96  }
97  //<Slider Callback !Start!>
98  //<Slider Callback !End!>
99  //<Tick Callback !Start!>
100
101 #endif
```

Listing E.13: include/components/GUI/GUISlice_references_content.hpp

GUISlice_screen.hpp

```
1  // ***********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ***********************************************
10
11 #ifndef _GUISLICE_SCREEN_HPP
12 #define _GUISLICE_SCREEN_HPP
13
14 #include <SPI.h>
15 #include <TFT_eSPI.h>      // Hardware-specific library
16 #include <string.h>
17 #include "components/settings.hpp"
18
19 namespace UVO_Components {
```

```
20 #define BACKLIGHT_PIN 12
21
22 namespace GUISlice {
23   #include "components/GUI/GUISliceBuilder_GSLC.hpp"
24
25   //TODO make screenstate into a class
26   //TODO MAKE STRUCT WITH elem id, whether it is editable, and pointer to
       what it changes, and which function to use for changing the value
27   struct s_screenState {
28
29     #define SETUP_PAGE_SELECTABLE_ITEMS_NUM 8
30     const int SETUP_page_selectable_items_size =
      SETUP_PAGE_SELECTABLE_ITEMS_NUM;
31     gslc_tsElemRef* SETUP_page_selectable_items[
      SETUP_PAGE_SELECTABLE_ITEMS_NUM];
32
33     // Must call this after initializing GSLC
34     void init_SETUP_sel_array(void){
35       SETUP_page_selectable_items[0] = m_pElem_SETUP_Intensity_255nm;
36       SETUP_page_selectable_items[1] = m_pElem_SETUP_Intensity_275nm;
37       SETUP_page_selectable_items[2] = m_pElem_SETUP_Intensity_285nm;
38       SETUP_page_selectable_items[3] = m_pElem_SETUP_Intensity_395nm;
39       SETUP_page_selectable_items[4] = m_pElem_SETUP_Hours;
40       SETUP_page_selectable_items[5] = m_pElem_SETUP_Minutes;
41       SETUP_page_selectable_items[6] = m_pElem_SETUP_Seconds;
42       SETUP_page_selectable_items[7] = m_pElem_SETUP_MotorIntensity;
43     }
44
45     gslc_tsElemRef** page_vec[1] = {SETUP_page_selectable_items};
46     int page_vec_array_sizes[1] = {SETUP_PAGE_SELECTABLE_ITEMS_NUM};
47
48     int current_page_idx = 0;
49     bool update_current_page = false;
50
51     int current_elem_idx = 0;
52
53     //TODO better name
54     bool elem_is_editing = false;
55
56
57     gslc_tsElemRef* getCurrentlySelectedElem(void){
58       return page_vec[current_page_idx][current_elem_idx];
59     }
60
61     uint16_t getCurrentlySelectedElemID(void){
62       return page_vec[current_page_idx][current_elem_idx]->pElem->nId;
63     }
64
65   };
66
67   class Screen{
68   public:
69     Screen(s_setupSettings* t_Settings);
70     Screen(void);
71     ~Screen();
72
```

```
73      // Both with and without setupSettings so that you can restart the
        module without having to change the settings
74      void init(void);
75      void init(s_setupSettings* t_initSettings);
76
77      void update(void);
78      void setSetupSettings(s_setupSettings* t_Settings);
79
80      void selectPreviousElem(void);
81      void selectNextElem(void);
82      void beginEditSelectedElem(void);
83      void endEditSelectedElem(void);
84      void toggleEditSelectedElem(void);
85
86      bool isEditingElement(void);
87      uint16_t getCurrentElementID(void);
88
89    private:
90      s_setupSettings* m_referenceSetupSettingsPointer;
91      s_setupSettings m_currentlyDisplayedSetupSettings;
92
93      s_screenState m_screenState;
94
95      const gslc_tsColor UVO_BLACK = {0, 0, 5};
96      const gslc_tsColor UVO_DARK_BLUE = {59, 51, 85};
97      const gslc_tsColor UVO_PURPLE = {93, 93, 129};
98      const gslc_tsColor UVO_LIGHT_BLUE = {191, 205, 224};
99      const gslc_tsColor UVO_WHITE = {254, 252, 253};
100
101     const int m_TFT_WIDTH = 480;
102     const int m_TFT_HEIGHT = 320;
103
104     // Need to swap the width and height due to the rotation of the tft
        screen
105     // Probably a bug in the system, or because I did something wrong
106     const int m_SPRITE_WIDTH = m_TFT_HEIGHT;
107     const int m_SPRITE_HEIGHT = m_TFT_WIDTH;
108
109     bool m_updateFrame;
110     u_int8_t m_Count = 0;
111
112     void writeFrame(void);
113
114     void setBackground(TFT_eSprite& sprite);
115     void setBackgroundText(TFT_eSPI& sprite);
116     void test(void);
117
118     void GUISliceInit(void);
119
120     void setColorFrame(gslc_tsElemRef* t_pElem, gslc_tsColor t_colFrame);
121     void setColorFill(gslc_tsElemRef* t_pElem, gslc_tsColor t_colFill);
122
123
124     //TODO better name
125     void resetElemOptions(gslc_tsElemRef* t_pElem);
126
```

```
127
128     // TODO rename the setSelected and updatedSelected. It is quite
        unclear.
129     void setSelectedElem(gslc_tsElemRef* t_pElem);
130     void setSelectedElem(int t_index);
131     void updateSelectedElem(int t_newSelectedIdx);
132
133     void displayAsSelected(gslc_tsElemRef* t_pElem);
134     void displayAsEditing(gslc_tsElemRef* t_pElem);
135
136     bool hasFillEnabled(gslc_tsElemRef* t_pElem);
137
138
139     void displayInteger(gslc_tsElemRef* t_pElem, uint8_t t_value);
140
141
142     void displaySetupSettings(s_setupSettings* t_new);
143     void displayIntensity(gslc_tsElemRef* t_pElem, uint8_t intensity);
144     void displayTime(gslc_tsElemRef* t_hour, gslc_tsElemRef* t_minutes,
        gslc_tsElemRef* t_seconds, time_t t_displayTime);
145
146     int uint8_to_percentage(uint8_t value);
147     template<typename T> std::optional<int> getIndex(T* t_vec, int t_size,
        T t_elem);
148   };
149 }
150 }
151 #endif
```

Listing E.14: include/components/GUI/GUISlice_screen.hpp

rotary_encoder.hpp

```
1  // ***********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ***********************************************
10
11 #ifndef _ROTARY_ENCODER_HPP
12 #define _ROTARY_ENCODER_HPP
13
14 #include <ESPRotary.h>
15
16 #endif
```

Listing E.15: include/components/GUI/rotary_encoder.hpp

screen.hpp

```
1  // ***********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
```

```
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#ifndef _SCREEN_HPP
#define _SCREEN_HPP

#include <SPI.h>
#include <TFT_eSPI.h>      // Hardware-specific library
#include <string.h>

#define BACKLIGHT_PIN 12

namespace UVO_Components {
namespace Screen {

  class Screen{
  private:
    TFT_eSPI m_tft = TFT_eSPI();
    TFT_eSprite m_screen = TFT_eSprite(&m_tft);
    // TFT_eSprite m_popupSprite = TFT_eSprite(&m_tft);

    const uint16_t UVO_BLACK = m_tft.color565(0, 0, 5);
    const uint16_t UVO_DARK_BLUE = m_tft.color565(59, 51, 85);
    const uint16_t UVO_PURPLE = m_tft.color565(93, 93, 129);
    const uint16_t UVO_LIGHT_BLUE = m_tft.color565(191, 205, 224);
    const uint16_t UVO_WHITE = m_tft.color565(254, 252, 253);

    const int m_TFT_WIDTH = 480;
    const int m_TFT_HEIGHT = 320;

    // Need to swap the width and height due to the rotation of the tft
    screen
    // Probably a bug in the system, or because I did something wrong
    const int m_SPRITE_WIDTH = m_TFT_HEIGHT;
    const int m_SPRITE_HEIGHT = m_TFT_WIDTH;

    bool m_updateFrame;

    void writeFrame(void);

    void setBackground(TFT_eSprite& sprite);
    void setBackgroundText(TFT_eSPI& sprite);
    void test(void);

  public:
    Screen(void);
    ~Screen();
    void init(void);
    void update(void);

  };
}
}
```

```
#endif
```

Listing E.16: include/components/GUI/screen.hpp

include/modules/control_module

main_controller.hpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#ifndef _MAIN_CONTROLLER_HPP
#define _MAIN_CONTROLLER_HPP

#include "main_controller_defines.hpp"

#ifdef USE_NORMAL_SCREEN
  #include "components/GUI/screen.hpp"
#else
  #include "components/GUI/GUISlice_screen.hpp"
#endif

#include "components/settings.hpp"
#include "components/communication/main_controller_communication_interface
    .hpp"
#include "components/GUI/button.hpp"
#include "components/GUI/rotary_encoder.hpp"

namespace UVO_MainController {

#ifdef USE_BUTTONS
enum UIEvent {UIbuttonUpPressed, UIbuttonDownPressed,
    UIbuttonRotaryPressed, UIrotaryRight, UIrotaryLeft, UInoEvent};
extern volatile UIEvent last_ui_event;

void onButtonUpPressISR(Button2& t_button);
void onButtonDownPressISR(Button2& t_button);
void onButtonRotaryPressISR(Button2& t_button);
void onRotaryRightISR(ESPRotary& t_rotary);
void onRotaryLeftISR(ESPRotary& t_rotary);
#endif


class MainController{
private:
  UVO_Components::s_systemState m_systemState;
  UVO_Components::s_setupSettings m_setupSettings;

  #ifdef USE_BUTTONS
  Button2 m_upButton;
  Button2 m_downButton;
```

```
49    Button2 m_rotaryButton;
50    ESPRotary m_rotaryEncoder;
51    #endif
52
53    #ifdef USE_SCREEN
54    #ifdef USE_NORMAL_SCREEN
55      #error "THE USE OF THE NORMAL SCREEN IS CURRENTLY NOT IMPLEMENTED,
      PLEASE USE THE GUISLICE CODE"
56      UVO_Components::Screen::Screen m_screen;
57    #else
58      UVO_Components::GUISlice::Screen m_screen;
59    #endif
60    #endif
61
62    #ifdef USE_COMMUNICATION_INTERFACE
63      UVO_MainController::MainCommunicationInterface
      m_communication_interface{UVO_CommunicationProtocol::
      MAIN_CONTROLLER_ADDRESS};
64    #endif
65
66    #ifdef USE_BUTTONS
67    void initUI(void);
68    void processUI(void);
69
70    void onButtonUpPress(Button2& t_button);
71    void onButtonDownPress(Button2& t_button);
72    void onEnterButtonPress(Button2& t_button);
73    void onRotaryRight(ESPRotary& t_rotary);
74    void onRotaryLeft(ESPRotary& t_rotary);
75    void changeSettingElement(int t_amount);
76    #endif
77
78  public:
79    MainController();
80    ~MainController();
81    void init(void);
82    void update(void);
83  };
84
85  }
86
87  #endif
```

Listing E.17: include/modules/control_module/main_controller.hpp

main_controller_defines.hpp

```
1  // ********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ********************************************
10
11 #ifndef _MAIN_CONTROLLER_DEFINES_HPP
```

```
12 #define _MAIN_CONTROLLER_DEFINES_HPP
13
14
15
16 // ********************************************************
17 //
18 //         TFT SCREEN DEFINES
19 //
20 // ********************************************************
21
22 #ifdef USE_SCREEN
23
24 // These values are defined in the platformio.ini, otherwise the code does
       not function, however, here is a list of the pin defines.
25
26 // #define USER_SETUP_LOADED
27 // #define ST7796_DRIVER
28
29 // #define TFT_RST 32
30 // #define TFT_MISO 19
31 // #define TFT_MOSI 23
32 // #define TFT_SCLK 18
33 // #define TFT_CS 4
34 // #define TFT_DC 2
35
36 // #define LOAD_GLCD
37 // #define LOAD_FONT2
38 // #define LOAD_FONT4
39 // #define LOAD_FONT6
40 // #define LOAD_FONT7
41 // #define LOAD_FONT8
42 // #define LOAD_GFXFF
43 // #define SMOOTH_FONT
44 // #define SPI_FREQUENCY 27000000
45
46 #endif
47
48
49 #define DTR_PIN 0
50 #define SPI_CS_SD 5
51 #define POWER_ENABLE_PIN 21
52
53 //TODO
54 // #define MOSI?
55 // #define MISO?
56 // #define SPI_CS_DISPLAY
57 // #define SCREEN_RST
58 // #define BACKLIT 12
59
60 #define ENABLE_SWITCH_PIN 22
61
62
63
64 // ********************************************************
65 //
66 //         BUTTONS AND ROTARY ENCODER DEFINES
```

```
//
// ******************************************************

// https://www.esp32.com/viewtopic.php?t=3206
#define SENSOR_VP 36
#define SENSOR_VN 39

#define ROTARY_ENCODER_A_PIN SENSOR_VP
#define ROTARY_ENCODER_B_PIN SENSOR_VN
#define ROTARY_ENCODER_PUSH_PIN 27

#define BUTTON_UP_PIN 34
#define BUTTON_DOWN_PIN 35



#define SPI_CS_XPT 33

#define nRESET_SWITCH_PIN 25
#define nRESET_LED_SWITCH_PIN 13
#define nRESET_MC_SWITCH 15

#define SDA_PIN 26
#define SCL_PIN 14


#endif
```

Listing E.18: include/modules/control_module/main_controller_defines.hpp

src

logger.cpp

```
// ********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ********************************************


//TODO
// log errors when a flag gets set high
//
```

Listing E.19: src/logger.cpp

main.cpp

```
// ********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
```

```
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#include "main.hpp"

void setup() {
  Serial.begin(9600);
  Controller.init();
}


void loop(void) {
  Controller.update();
}
```

Listing E.20: src/main.cpp

src/communication_test

I2C_communication_test.cpp

```
// ***********************************************
//    EE3L11: Bachelor Graduation Project
//    GROUP M: UVC SEED STERILIZATION
//    SUBGROUP: SOFTWARE AND CONTROL
//    MEMBERS: Erman Erg l , Erik van Weelderen
//
//    BY ERIK VAN WEELDEREN
//    DATE: 16-6-2023
// ***********************************************

#include "I2C_communication_test.hpp"
#include "components/communication/communication_protocol.hpp"

namespace UVO_UNIT_TESTS {
  void receive(int);
  void request(void);

  UNOTransceiver::UNOTransceiver(OperationMode t_operationMode, int
    t_I2C_address){
    m_operationMode = t_operationMode;
    m_I2C_address = t_I2C_address;
  }

  void UNOTransceiver::init(){
    slaveInit();
    Wire.begin(m_I2C_address);
    Serial.begin(9600);  // start serial for output
  }

  void UNOTransceiver::slaveInit(){
    // Wire.onRequest(requestEvent);
    Wire.onReceive(receive);
    Wire.onRequest(request);
  }

```

```
35
36  void UNOTransceiver::update(){
37    slaveUpdate();
38  }
39
40  void receive(int){
41    UVO_CommunicationProtocol::TPackageTypeToken packageType;
42    packageType = (UVO_CommunicationProtocol::TPackageTypeToken) Wire.read
      ();
43    Serial.println(packageType);
44
45    while(Wire.available()){
46      Serial.println(Wire.read());
47    }
48  }
49
50  void request(void){
51    double test = 1.5f;
52    byte* tp = (byte*) &test;
53    Wire.write(tp, sizeof(test));
54  }
55
56  void UNOTransceiver::slaveUpdate(void){
57    delay(100);
58  }
59 }
```

Listing E.21: src/communication_test/I2C_communication_test.cpp

I2C_communication_test.hpp

```
1  // ********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ********************************************
10
11 #ifndef _UNO_CONTROLLERS_HPP
12 #define _UNO_CONTROLLERS_HPP
13
14 #include "components/communication/I2C_interface.hpp"
15 #include "components/communication/main_controller_communication_interface
       .hpp"
16 #include "components/communication/communication_protocol.hpp"
17 // #include <random.h>
18
19
20 namespace UVO_UNIT_TESTS {
21   const int MASTER_ADDRESS = 40;
22   const int SLAVE_ADDRESS = 50;
23
24   enum OperationMode {SLAVE_OPERATION_MODE, MASTER_OPERATION_MODE};
25
26   class UNOTransceiver
```

```
27   {
28   private:
29     OperationMode m_operationMode;
30     int m_I2C_address;
31     void masterInit(void);
32     void slaveInit(void);
33
34     void masterUpdate(void);
35     void slaveUpdate(void);
36
37   public:
38     UNOTransceiver(OperationMode t_operationMode, int t_I2C_address);
39     void init();
40     void update();
41   };
42
43   void requestEvent(void);
44
45 }
46
47 #endif
```

Listing E.22: src/communication_test/I2C_communication_test.hpp

src/components

sd_interface.cpp

```
1  // ***********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ***********************************************
10
11 class SDInterface
12 {
13 private:
14   /* data */
15 public:
16   SDInterface(/* args */);
17   ~SDInterface();
18 };
19
20 SDInterface::SDInterface(/* args */)
21 {
22 }
23
24 SDInterface::~SDInterface()
25 {
26 }
```

Listing E.23: src/components/sd_interface.cpp

sensor.cpp

```cpp
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#include "components/communication/sensor.hpp"
#include <stdio.h>

namespace UVO_Components{

namespace sensors {


}
}
```

Listing E.24: src/components/sensor.cpp

settings.cpp

```cpp
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#include "components/settings.hpp"
```

Listing E.25: src/components/settings.cpp

src/components/communication

I2C_interface.cpp

```cpp
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#include "components/communication/I2C_interface.hpp"

// https://www.gammon.com.au/i2c


namespace UVO_Components {
```

```
17
18 I2CInterface::I2CInterface(int t_I2C_address){
19   m_I2C_address = t_I2C_address;
20
21   Wire.begin(m_I2C_address);
22 }
23
24 // I2CInterface::I2CInterface(int t_I2C_address, int t_SDA_pin, int
       t_SCL_pin){
25 //   m_SDA_pin = t_SDA_pin;
26 //   m_SCL_pin = t_SCL_pin;
27 //   m_I2C_address = t_I2C_address;
28
29 //   // Wire.begin(m_SDA_pin, m_SCL_pin, m_I2C_address);
30 // }
31
32
33 I2CInterface::~I2CInterface(){
34   Wire.end();
35 }
36
37 void I2CInterface::sendMessages(int address, byte* message, int
       message_length){
38   Wire.beginTransmission(address);
39   Wire.write(message, (size_t) message_length);
40   Wire.endTransmission();
41
42   //TODO fix error checking
43   // return true;
44 }
45
46 int I2CInterface::requestAndReadAnswer(int I2C_address, byte*
       receive_message, int bytes_requested){
47   //TODO CHECK OUT IF requestFrom can also be implemented using restart=
       false
48   int num_bytes_received = Wire.requestFrom(I2C_address, bytes_requested);
49
50   //TODO MULTILPE CHECKS
51   if(Wire.available()){
52     Wire.readBytes(receive_message, num_bytes_received);
53   }
54
55   //TODO RETURN OPTIONAL VALUE IF ERROR FOR EXAMPLE
56   return num_bytes_received;
57 }
58
59 void I2CInterface::onRequest( void (*t_function)(void) ){
60   Wire.onRequest(t_function);
61 }
62
63 void I2CInterface::onReceive( void (*t_function)(int) ){
64   Wire.onReceive(t_function);
65 }
66
67 }
```

Listing E.26: src/components/communication/I2C_interface.cpp

main_controller_communication_interface.cpp

```cpp
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#include "components/communication/main_controller_communication_interface.hpp"

namespace UVO_MainController {

MainCommunicationInterface::MainCommunicationInterface(int t_I2C_address)
  : m_I2C_Interface(t_I2C_address)
{

}

MainCommunicationInterface::~MainCommunicationInterface(){

}

void MainCommunicationInterface::init(void){

}


// TODO: IMPLEMENT https://forum.arduino.cc/t/how-to-properly-use-wire-onreceive/891195/2
void MainCommunicationInterface::update(void){
  //TODO Implement good update function
  double received_data;

  received_data = requestSensorData(UVO_CommunicationProtocol::sensors::TOP_LEDDriver::current_sensor_255nm);

  Serial.println(received_data);

  delay(1000);
}

int MainCommunicationInterface::sendMessageAndReadResponse(int t_I2C_slave_address, byte* t_message, int t_message_length, int t_bytes_requested, byte* t_response_data){
  int response_length;

  m_I2C_Interface.sendMessages(t_I2C_slave_address, t_message, t_message_length);

  response_length = m_I2C_Interface.requestAndReadAnswer(t_I2C_slave_address, t_response_data, t_bytes_requested);

```

```
49    if (response_length != t_bytes_requested){
50      //TODO Add log here.
51    }
52
53    return response_length;
54  }
55
56  double MainCommunicationInterface::requestSensorData(
      UVO_CommunicationProtocol::Sensor t_sensor){
57
58    double response = 0;
59    int bytes_requested = sizeof(response);
60    byte* response_pointer = (byte*) &response;
61
62    int received_length = 0;
63
64    int I2C_address = t_sensor.module_address_I2C;
65
66    byte message[] = {UVO_CommunicationProtocol::PackageTypeToken::
       REQUEST_SENSOR_DATA, t_sensor.sensorToken};
67    int message_length = sizeof(message) / sizeof(message[0]);
68
69    received_length = sendMessageAndReadResponse(I2C_address, message,
       message_length, bytes_requested, response_pointer);
70
71    if (received_length != bytes_requested){
72      // TODO LOG SOMETHING IS WRONG
73    }
74
75
76    return response;
77  }
78
79  //bool MainCommunicationInterface::setPWMDutyCycle(Driver driver, int pwm)
      {
80
81  // }
82
83  }
```

Listing E.27: src/components/communication/main_controller_communication_interface.cpp

src/components/GUI

button.cpp

```
1  // ********************************************
2  //    EE3L11: Bachelor Graduation Project
3  //    GROUP M: UVC SEED STERILIZATION
4  //    SUBGROUP: SOFTWARE AND CONTROL
5  //    MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //    BY ERIK VAN WEELDEREN
8  //    DATE: 16-6-2023
9  // ********************************************
10
11 #include "components/GUI/button.hpp"
```

Listing E.28: src/components/GUI/button.cpp

GUISlice_screen.cpp

```cpp
// ***********************************************
//     EE3L11: Bachelor Graduation Project
//     GROUP M: UVC SEED STERILIZATION
//     SUBGROUP: SOFTWARE AND CONTROL
//     MEMBERS: Erman Erg l , Erik van Weelderen
//
//     BY ERIK VAN WEELDEREN
//     DATE: 16-6-2023
// ***********************************************

#include "components/GUI/GUISlice_screen.hpp"

namespace UVO_Components {
namespace GUISlice {

//TODO include logo onto screen

#include "components/GUI/GUISlice_references_content.hpp"

Screen::Screen(s_setupSettings* t_initSettings){
  init(t_initSettings);
}

Screen::Screen(void){
  init();
}

Screen::~Screen(){

}

void Screen::GUISliceInit(void){
  gslc_InitDebug(&DebugOut);

  InitGUIslice_gen();
  m_screenState.init_SETUP_sel_array();
}

void Screen::init(void){
  pinMode(BACKLIGHT_PIN, OUTPUT);
  digitalWrite(BACKLIGHT_PIN, HIGH);

  m_screenState.elem_is_editing = false;
  m_screenState.current_elem_idx = 0;
  m_screenState.current_page_idx = 0;

  GUISliceInit();
}

void Screen::init(s_setupSettings* t_initSettings){
  setSetupSettings(t_initSettings);
  init();
}

void Screen::setSetupSettings(s_setupSettings* t_Settings){
```

```
56    m_referenceSetupSettingsPointer = t_Settings;
57    displaySetupSettings(m_referenceSetupSettingsPointer);
58    gslc_Update(&m_gui);
59  }
60
61
62  void Screen::update(void){
63    if (m_referenceSetupSettingsPointer->isUpdated){
64      displaySetupSettings(m_referenceSetupSettingsPointer);
65      m_referenceSetupSettingsPointer->isUpdated = false;
66    }
67
68    gslc_Update(&m_gui);
69  }
70
71
72
73  void Screen::displaySetupSettings(s_setupSettings* t_new){
74    //TODO write this more elligible
75
76    //TODO IMPLEMENT SWITCH STATEMENT FOR WHICH PAGE YOU ARE ON
77
78    if (m_currentlyDisplayedSetupSettings.LED_intensity_255nm != t_new->
       LED_intensity_255nm){
79      m_currentlyDisplayedSetupSettings.LED_intensity_255nm = t_new->
       LED_intensity_255nm;
80      displayIntensity(m_pElem_SETUP_Intensity_255nm,
       m_currentlyDisplayedSetupSettings.LED_intensity_255nm);
81    }
82
83    if (m_currentlyDisplayedSetupSettings.LED_intensity_275nm != t_new->
       LED_intensity_275nm){
84      m_currentlyDisplayedSetupSettings.LED_intensity_275nm = t_new->
       LED_intensity_275nm;
85      displayIntensity(m_pElem_SETUP_Intensity_275nm,
       m_currentlyDisplayedSetupSettings.LED_intensity_275nm);
86    }
87
88    if (m_currentlyDisplayedSetupSettings.LED_intensity_285nm != t_new->
       LED_intensity_285nm){
89      m_currentlyDisplayedSetupSettings.LED_intensity_285nm = t_new->
       LED_intensity_285nm;
90      displayIntensity(m_pElem_SETUP_Intensity_285nm,
       m_currentlyDisplayedSetupSettings.LED_intensity_285nm);
91    }
92
93    if (m_currentlyDisplayedSetupSettings.LED_intensity_395nm != t_new->
       LED_intensity_395nm){
94      m_currentlyDisplayedSetupSettings.LED_intensity_395nm = t_new->
       LED_intensity_395nm;
95      displayIntensity(m_pElem_SETUP_Intensity_395nm,
       m_currentlyDisplayedSetupSettings.LED_intensity_395nm);
96    }
97
98    if (m_currentlyDisplayedSetupSettings.motor_intensity != t_new->
       motor_intensity){
```

```
99      m_currentlyDisplayedSetupSettings.motor_intensity = t_new->
        motor_intensity;
100     displayIntensity(m_pElem_SETUP_MotorIntensity,
        m_currentlyDisplayedSetupSettings.motor_intensity);
101   }
102
103   if (m_currentlyDisplayedSetupSettings.targetExposureTime != t_new->
        targetExposureTime){
104     m_currentlyDisplayedSetupSettings.targetExposureTime = t_new->
        targetExposureTime;
105     displayTime(m_pElem_SETUP_Hours, m_pElem_SETUP_Minutes,
        m_pElem_SETUP_Seconds, m_currentlyDisplayedSetupSettings.
        targetExposureTime);
106   }
107
108 }
109
110 void Screen::displayInteger(gslc_tsElemRef* t_pElem, uint8_t t_value){
111   char txt[4];
112   snprintf(txt, 4, "%02d", t_value);
113   gslc_ElemSetTxtStr(&m_gui, t_pElem, txt);
114 }
115
116 void Screen::displayIntensity(gslc_tsElemRef* t_pElem, uint8_t t_intensity
        ){
117   int intensity_percentage = uint8_to_percentage(t_intensity);
118   displayInteger(t_pElem, intensity_percentage);
119 }
120
121 void Screen::displayTime(gslc_tsElemRef* t_pElem_hour, gslc_tsElemRef*
        t_pElem_minutes, gslc_tsElemRef* t_pElem_seconds, time_t t_displayTime)
        {
122   char hours_txt[3] = {0};
123   char minutes_txt[3] = {0};
124   char seconds_txt[3] = {0};
125   tm* curr_tm = localtime(&t_displayTime);
126   strftime(hours_txt, sizeof(hours_txt), "%HH", curr_tm);
127   strftime(minutes_txt, sizeof(minutes_txt), "%MM", curr_tm);
128   strftime(seconds_txt, sizeof(seconds_txt), "%SS", curr_tm);
129
130   gslc_ElemSetTxtStr(&m_gui, t_pElem_hour, hours_txt);
131   gslc_ElemSetTxtStr(&m_gui, t_pElem_minutes, minutes_txt);
132   gslc_ElemSetTxtStr(&m_gui, t_pElem_seconds, seconds_txt);
133 }
134
135
136 void Screen::selectPreviousElem(void){
137   int previous_idx = m_screenState.current_elem_idx - 1;
138   int min_idx = 0;
139
140   previous_idx = (previous_idx < min_idx) ? min_idx : previous_idx;
141
142   setSelectedElem(previous_idx);
143 }
144
145 void Screen::selectNextElem(void){
```

```
146   gslc_tsElemRef** current_elem_array = m_screenState.page_vec[
        m_screenState.current_page_idx];
147   int max_idx = m_screenState.page_vec_array_sizes[m_screenState.
        current_page_idx] - 1;
148
149   int next_idx = m_screenState.current_elem_idx + 1;
150   next_idx = (next_idx > max_idx) ? max_idx : next_idx;
151
152   setSelectedElem(next_idx);
153 }
154
155 void Screen::endEditSelectedElem(void){
156   m_screenState.elem_is_editing = false;
157   gslc_tsElemRef* current_elem = m_screenState.getCurrentlySelectedElem();
158   displayAsSelected(current_elem);
159 }
160
161 void Screen::beginEditSelectedElem(void){
162   m_screenState.elem_is_editing = true;
163   gslc_tsElemRef* current_elem = m_screenState.getCurrentlySelectedElem();
164   displayAsEditing(current_elem);
165 }
166
167 void Screen::toggleEditSelectedElem(void){
168   if (m_screenState.elem_is_editing){
169     endEditSelectedElem();
170   }
171   else{
172     beginEditSelectedElem();
173   }
174 }
175
176 bool Screen::isEditingElement(void){
177   return m_screenState.elem_is_editing;
178 }
179
180 uint16_t Screen::getCurrentElementID(void){
181   return m_screenState.getCurrentlySelectedElemID();
182 }
183
184
185 void Screen::setSelectedElem(gslc_tsElemRef* t_pElem){
186   std::optional<int> index = getIndex(m_screenState.
        SETUP_page_selectable_items, m_screenState.
        SETUP_page_selectable_items_size, t_pElem);
187
188   if (!index){
189     //TODO FIX WITH CUSTOM DEBUGGING FUNCTION
190     throw "Element not in array";
191   }
192
193   updateSelectedElem(index.value());
194 }
195
196 void Screen::setSelectedElem(int t_index){
197   updateSelectedElem(t_index);
```

```
198 }
199
200 void Screen::updateSelectedElem(int t_newSelectedIdx){
201   m_screenState.elem_is_editing = false;
202   int current_elem_idx = m_screenState.current_elem_idx;
203
204   if (t_newSelectedIdx != current_elem_idx){
205     gslc_tsElemRef* selectedElem = m_screenState.getCurrentlySelectedElem
      ();
206     resetElemOptions(selectedElem);
207
208     m_screenState.current_elem_idx = t_newSelectedIdx;
209
210     selectedElem = m_screenState.getCurrentlySelectedElem();
211     displayAsSelected(selectedElem);
212   }
213
214 }
215
216 void Screen::setColorFrame(gslc_tsElemRef* t_pElem, gslc_tsColor
      t_colFrame){
217   gslc_tsColor current_colFill =  t_pElem->pElem->colElemFill;
218   gslc_tsColor current_colFillGlow = t_pElem->pElem->colElemFillGlow;
219   gslc_ElemSetCol(&m_gui, t_pElem, t_colFrame, current_colFill,
      current_colFillGlow);
220 }
221
222 void Screen::setColorFill(gslc_tsElemRef* t_pElem, gslc_tsColor t_colFill)
      {
223   gslc_tsColor current_colFrame = t_pElem->pElem->colElemFrame;
224   gslc_tsColor current_colFillGlow = t_pElem->pElem->colElemFillGlow;
225
226   gslc_ElemSetCol(&m_gui, t_pElem, current_colFrame, t_colFill,
      current_colFillGlow);
227 }
228
229 void Screen::resetElemOptions(gslc_tsElemRef* t_pElem){
230   gslc_ElemSetFrameEn(&m_gui, t_pElem, false);
231   gslc_ElemSetFillEn(&m_gui, t_pElem, false);
232   gslc_ElemSetTxtCol(&m_gui, t_pElem, UVO_BLACK);
233 }
234
235
236
237 void Screen::displayAsSelected(gslc_tsElemRef* t_pElem){
238   resetElemOptions(t_pElem);
239   gslc_ElemSetFrameEn(&m_gui, t_pElem, true);
240 }
241
242 void Screen::displayAsEditing(gslc_tsElemRef* t_pElem){
243   resetElemOptions(t_pElem);
244
245   gslc_ElemSetFrameEn(&m_gui, t_pElem, true);
246   gslc_ElemSetFillEn(&m_gui, t_pElem, true);
247   gslc_ElemSetTxtCol(&m_gui, t_pElem, UVO_WHITE);
248 }
```

```
249
250
251 bool Screen::hasFillEnabled(gslc_tsElemRef* t_pElem){
252   return t_pElem->pElem->nFeatures & (1 << GSLC_ELEM_FEA_FILL_EN);
253 }
254
255
256 //TODO MOVE TO OTHER FILE
257 int Screen::uint8_to_percentage(uint8_t value){
258   return value * 100.0/255;
259 }
260
261 //TODO MOVE TO OTHER FILE
262 template<typename T>
263 std::optional<int> Screen::getIndex(T* t_vec, int t_size, T t_elem){
264   for (int i = 0; i < t_size; i++){
265     if (t_vec[i] == t_elem){
266       return i;
267     }
268   }
269
270   return std::nullopt;
271 }
272
273 // int Screen::clip(int t_val, int t_min, int t_max){
274 //     // int idx = (t_index > max_idx) ? max_idx : t_index;
275 //  // idx = (idx < 0) ? 0 : idx;
276
277 // }
278
279 }
280 }
```

Listing E.29: src/components/GUI/GUISlice_screen.cpp

screen.cpp

```
1  // ***********************************************
2  //     EE3L11: Bachelor Graduation Project
3  //     GROUP M: UVC SEED STERILIZATION
4  //     SUBGROUP: SOFTWARE AND CONTROL
5  //     MEMBERS: Erman Erg l , Erik van Weelderen
6  //
7  //     BY ERIK VAN WEELDEREN
8  //     DATE: 16-6-2023
9  // ***********************************************
10
11 #include "components/GUI/screen.hpp"
12
13 // #include <stdio.h>
14 // #include <iostream>
15
16 namespace UVO_Components {
17 namespace Screen {
18
19 Screen::Screen(void) {
20   init();
21 }
```

```
22
23 Screen::~Screen(){
24
25 }
26
27 void Screen::init(void){
28   m_tft.init();
29   m_tft.setRotation(1);
30
31   pinMode(BACKLIGHT_PIN, OUTPUT);
32   digitalWrite(BACKLIGHT_PIN, HIGH);
33
34   m_tft.setSwapBytes(true);                        // swap the byte order
       for pushImage() - corrects endianness
35   m_tft.fillScreen(TFT_SKYBLUE);
36   m_updateFrame = true;
37 }
38
39 void Screen::update(void){
40   if (m_updateFrame){
41     m_updateFrame = false;
42     test();
43   }
44 }
45
46
47 void Screen::test(void){
48   m_tft.fillScreen(UVO_DARK_BLUE);
49
50   m_screen.createSprite(m_SPRITE_WIDTH, m_SPRITE_HEIGHT);
51   m_screen.fillScreen(TFT_TRANSPARENT);
52
53   m_screen.fillRect(0, 0, 300, 100, TFT_GREEN);
54
55   setBackground(m_screen);
56
57   m_screen.setCursor(100, 300);
58   m_screen.setTextColor(TFT_BLACK, TFT_TRANSPARENT);
59   m_screen.setTextSize(3);
60   m_screen.println("Hello\n");
61
62   m_screen.pushSprite(0,0, TFT_TRANSPARENT);
63   m_screen.deleteSprite();
64 }
65
66 void Screen::setBackground(TFT_eSprite& sprite){
67   sprite.fillCircle(m_TFT_WIDTH/3, m_TFT_HEIGHT/2, 40, TFT_GREENYELLOW);
68   sprite.fillCircle(m_TFT_WIDTH/3, m_TFT_HEIGHT/2, 40, TFT_GREENYELLOW);
69 }
70
71 void Screen::setBackgroundText(TFT_eSPI& sprite){
72   sprite.setTextColor(TFT_BLACK);
73   sprite.setTextSize(3);
74
75   sprite.setCursor(20, 30);
76   sprite.println("Intensity (255nm):");
```

```
77
78    sprite.setCursor(20, 60);
79    sprite.println("Intensity (275nm):");
80
81    sprite.setCursor(20, 90);
82    sprite.println("Intensity (285nm):");
83
84    sprite.setCursor(20, 120);
85    sprite.println("Intensity (395nm):");
86  }
87
88  void Screen::writeFrame(void){
89    // m_tft.drawBitmap(0, 0, image, 100, 100, TFT_WHITE, TFT_BLACK);
90
91    m_tft.fillScreen(TFT_BLACK);
92
93    // m_tft.setRotation(3);
94    // m_tft.fillScreen (TFT_GREEN);
95    m_tft.fillScreen(TFT_BLUE);
96    // m_tft.fillRect(0,0, (int) (width*0.5), (int) (height*0.9), TFT_GREEN)
       ;
97
98    TFT_eSprite test = TFT_eSprite(&m_tft);
99
100   // test.createSprite(100, 100);
101   m_screen.createSprite(160, 160);
102   m_screen.setSwapBytes(true);                        // swap the byte order
        for pushImage() - corrects endianness
103
104   // Useful to debug
105   if (m_screen.created()){
106     m_tft.fillRect(0,0,50,50, TFT_GREEN);
107   }
108   else{
109     m_tft.fillRect(0,0,50,50, TFT_RED);
110   }
111
112   // m_screen.pushSprite(300,100);
113 }
114
115 }
116 }
```

Listing E.30: src/components/GUI/screen.cpp

src/modules

main_controller.cpp

```
1 // ********************************************
2 //    EE3L11: Bachelor Graduation Project
3 //    GROUP M: UVC SEED STERILIZATION
4 //    SUBGROUP: SOFTWARE AND CONTROL
5 //    MEMBERS: Erman Erg l , Erik van Weelderen
6 //
7 //    BY ERIK VAN WEELDEREN
8 //    DATE: 16-6-2023
9 // ********************************************
```

```
10
11 #include "modules/control_module/main_controller.hpp"
12
13 namespace UVO_MainController {
14   #ifdef USE_BUTTONS
15   volatile UIEvent last_ui_event = UInoEvent;
16   #endif
17
18   MainController::MainController(){
19
20   }
21
22   MainController::~MainController(){
23
24   }
25
26   void MainController::init(void){
27     #ifdef USE_SCREEN
28       m_screen.init(&m_setupSettings);
29     #endif
30
31     #ifdef USE_COMMUNICATION_INTERFACE
32       m_communication_interface.init();
33     #endif
34
35     // initUI();
36   }
37
38   void MainController::update(void){
39     if (!m_setupSettings.isUpdated){
40       delay(1000);
41       m_setupSettings.addSeconds(1);
42
43       m_setupSettings.isUpdated = true;
44
45       m_screen.toggleEditSelectedElem();
46     }
47
48     #ifdef USE_BUTTONS
49     if (last_ui_event != UInoEvent){
50       processUI();
51     }
52     #endif
53
54     #ifdef USE_SCREEN
55       m_screen.update();
56     #endif
57
58     #ifdef USE_COMMUNICATION_INTERFACE
59       m_communication_interface.update();
60     #endif
61
62   }
63
64   #ifdef USE_BUTTONS
65   void MainController::initUI(void){
```

```
66      //TODO CHECK IF 5U is correct
67      m_upButton.begin(BUTTON_UP_PIN, 5U, false);
68      m_downButton.begin(BUTTON_DOWN_PIN, 5U, false);
69      m_rotaryButton.begin(ROTARY_ENCODER_PUSH_PIN, 5U, false);
70      //TODO FIND STEPS PER INC
71      m_rotaryEncoder.begin(ROTARY_ENCODER_A_PIN, ROTARY_ENCODER_B_PIN);
72
73      m_upButton.setClickHandler(onButtonUpPressISR);
74      m_downButton.setClickHandler(onButtonDownPressISR);
75      m_rotaryButton.setClickHandler(onButtonRotaryPressISR);
76
77      m_rotaryEncoder.setRightRotationHandler(onRotaryRightISR);
78      m_rotaryEncoder.setLeftRotationHandler(onRotaryLeftISR);
79    }
80
81    void MainController::processUI(void){
82      switch (last_ui_event){
83      case UIbuttonUpPressed:
84        onButtonUpPress(m_upButton);
85        break;
86      case UIbuttonDownPressed:
87        onButtonDownPress(m_downButton);
88        break;
89      case UIbuttonRotaryPressed:
90        onButtonRotaryPressISR(m_rotaryButton);
91        break;
92      case UIrotaryLeft:
93        onRotaryLeft(m_rotaryEncoder);
94        break;
95      case UIrotaryRight:
96        onRotaryRight(m_rotaryEncoder);
97        break;
98
99      default:
100       break;
101     }
102
103     last_ui_event = UInoEvent;
104   }
105
106   void onButtonUpPressISR(Button2& t_button){
107     last_ui_event=UIbuttonUpPressed;
108   }
109   void onButtonDownPressISR(Button2& t_button){
110     last_ui_event=UIbuttonDownPressed;
111   }
112   void onButtonRotaryPressISR(Button2& t_button){
113     last_ui_event=UIbuttonRotaryPressed;
114   }
115   void onRotaryRightISR(ESPRotary& t_rotary){
116     last_ui_event=UIrotaryRight;
117   }
118   void onRotaryLeftISR(ESPRotary& t_rotary){
119     last_ui_event=UIrotaryLeft;
120   }
121
```

```
122  void MainController::onButtonUpPress(Button2& t_button){
123    if(!m_screen.isEditingElement()){
124      m_screen.selectPreviousElem();
125    }
126  }
127
128  void MainController::onButtonDownPress(Button2& t_button){
129    if(!m_screen.isEditingElement()){
130      m_screen.selectNextElem();
131    }
132  }
133
134  void MainController::onEnterButtonPress(Button2& t_button){
135    if(!m_screen.isEditingElement()){
136      m_screen.beginEditSelectedElem();
137    }
138    else{
139
140    }
141
142
143  }
144
145  void MainController::onRotaryRight(ESPRotary& t_rotary){
146    if(m_screen.isEditingElement()){
147      changeSettingElement(1);
148    }
149    else{
150      m_screen.selectNextElem();
151    }
152  }
153
154  void MainController::onRotaryLeft(ESPRotary& t_rotary){
155    if(m_screen.isEditingElement()){
156      changeSettingElement(-1);
157    }
158    else{
159      m_screen.selectPreviousElem();
160    }
161
162  }
163
164
165  void MainController::changeSettingElement(int t_amount){
166    int16_t elem_id = m_screen.getCurrentElementID();
167
168    switch(elem_id)
169    {
170    case UVO_Components::GUISlice::E_ELEM_SETUP_Intensity_255nm:
171      m_setupSettings->LED_intensity_255nm += t_amount;
172      break;
173    case UVO_Components::GUISlice::E_ELEM_SETUP_Intensity_275nm:
174      m_setupSettings->LED_intensity_275nm += t_amount;
175      break;
176    case UVO_Components::GUISlice::E_ELEM_SETUP_Intensity_285nm:
177      m_setupSettings->LED_intensity_285nm += t_amount;
```

```
178         break;
179       case UVO_Components::GUISlice::E_ELEM_SETUP_Intensity_395nm:
180         m_setupSettings->LED_intensity_395nm += t_amount;
181         break;
182       case UVO_Components::GUISlice::E_ELEM_SETUP_Hours:
183         m_setupSettings->addHours(t_amount);
184         break;
185       case UVO_Components::GUISlice::E_ELEM_SETUP_Minutes:
186         m_setupSettings->addMinutes(t_amount);
187         break;
188       case UVO_Components::GUISlice::E_ELEM_SETUP_Seconds:
189         m_setupSettings->addSeconds(t_amount);
190         break;
191       case UVO_Components::GUISlice::E_ELEM_SETUP_MotorIntensity:
192         m_setupSettings->motor_intensity += t_amount;
193         break;
194       default:
195         break;
196       }
197     }
198     #endif
199
200 }
```

Listing E.31: src/modules/main_controller.cpp

F

Testing Report

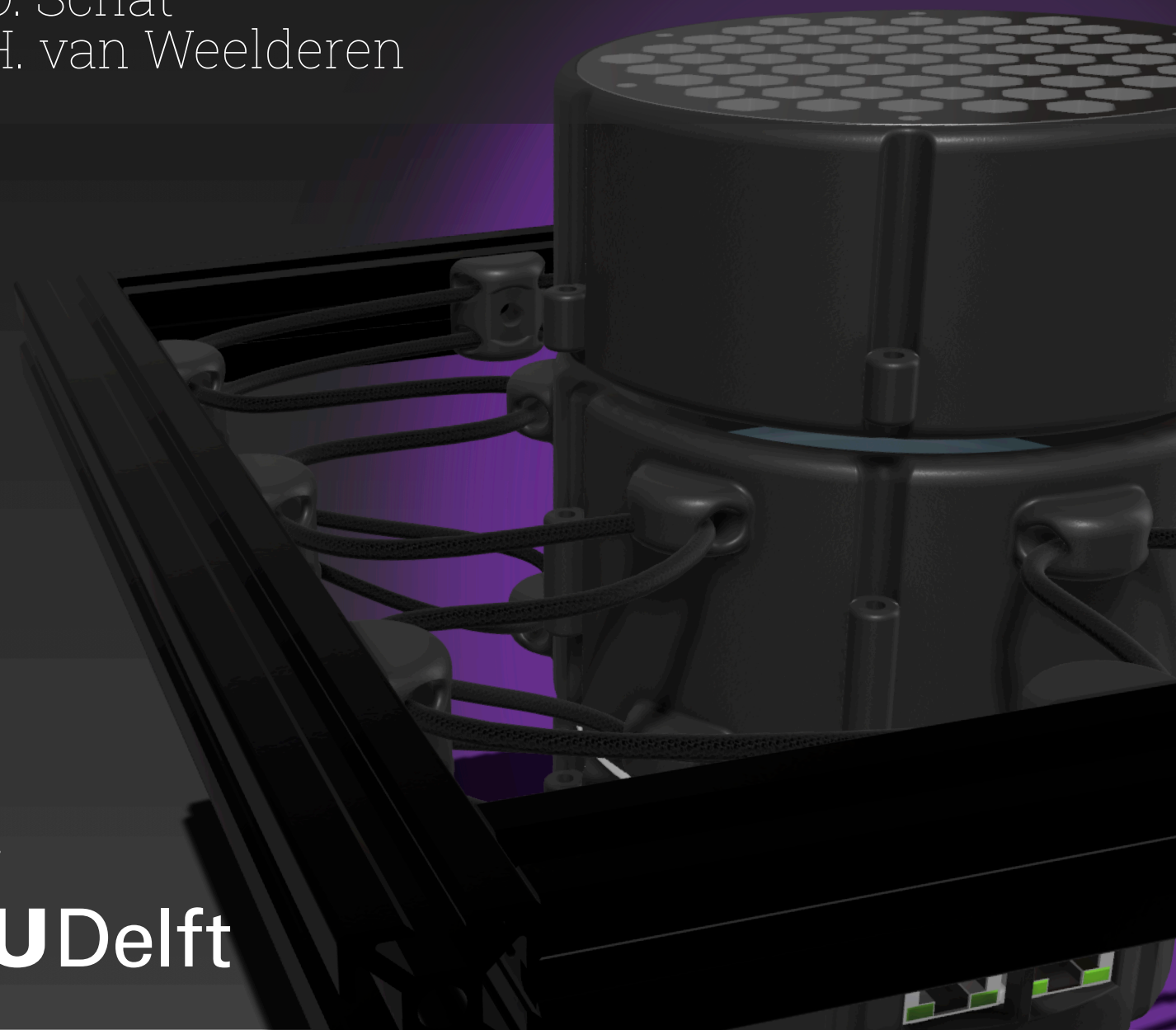
UV-C LED Seed Disinfection

Test results

EE3L11: Bachelor Graduation Project

E. Ergül
R.W.L. Imbens
L.C. Klootwijk
M. Mazurovs
D.O. Schat
E.H. van Weelderen

Delft University of Technology



TU Delft

Preface & acknowledgement

This report presents the experiments of the research project aimed at the validation of the custom-built UV-C LED disinfection device, targeted to disinfect fungi from seed. The project represents a collaborative effort by the team of students that set out to develop this device. Drawing upon the team's collective expertise in Electrical Engineering, there is designed and constructed a specialized UV-C LED device capable of irradiating seed with far-ultraviolet light, known to be capable of disinfection. The objective of this study was to evaluate the device's performance in seed disinfection and to test different setups in terms of wavelength to obtain meaningful results in hopefully eradicating pathogens, thereby enhancing seed quality.

Throughout the project, the testing protocol was implemented to assess the device's ability to disinfect cabbage seed. The team prepared and conducted controlled experiments, in which multiple variables such as exposure time, the wavelength of the UV-C light, and the intensity of UV-C radiation could be controlled.

We express our deepest gratitude to all those who contributed to this endeavour, including our team members, advisers, and the support of Rijk Zwaan who provided resources and guidance. It is our hope that this report will not only advance scientific knowledge but also inspire further exploration and adoption of UV-C using LED.

E. Ergül
R.W.L. Imbens
L.C. Klootwijk
M. Mazurovs
D.O. Schat
E.H. van Weelderen
Delft, June 2023

Abstract

This report gives the results of using UV-C LED light for disinfecting plant seeds. This study focuses on destroying Alternaria fungi on rape seeds. The machine used for the tests is a custom developed UV-C radiating device which is enclosed in a safe casing for user-safety and features an integrated control unit for easy selection of variables and parameters. Parameters that can be set are wavelength, intensity and dose. Also, there is a vibration motor built into the seed bed so that seeds can be turned around during testing to ensure an even radiation. The method used for testing is the PCR-method. The results of the tests on the seeds does not give a significant difference, so conclusions can not be determined yet. Other methods are needed to determine the effectiveness of the tests. Based on integration testing of the machine and the literature, the machine has the potential to successfully disinfect the seeds, so doing the recommended future work can be very interesting.

Nomenclature

Abbreviations

Abbreviation	Definition
LED	Light Emitting Diode
UVC & UV-C	Lowest range UV light (100 - 280 nm)
PSU	Power Supply Unit
CU	Control Unit
MCU & MC	Motor Control Unit
LDS	LED Driving & Sensing module
PCB	Printed Circuit board
IC	Integrated-Circuit, usually referring to a chip or module
PoR	programme of requirements
MR	Mandatory requirements
ToR	trade-off requirements

Symbols

Symbol	Definition	Unit
P	Power	[W]
D	Dose	[mJ/cm^2]
t	Time	[s]
λ	Wavelength	[nm]

Contents

1 Introduction

As the world population increases and food production should be more and more efficient and sustainable, improvements and offering more sustainable technologies in the agricultural sector become more and more important. Seed disinfection plays a critical role in ensuring the quality and health of crops. Contamination by pathogenic microorganisms, such as Alternaria, can significantly impact seed quality and crop yield. In the pursuit of effective seed disinfection methods, our research team has conducted a study to evaluate the efficacy of an in-house built UV-C device hopefully capable of eliminating microbial contaminants from seeds.

This technical report presents the findings of the disinfection from this UV-C device and an analysis of some key parameters. With the use of the far-ultraviolet irradiation principles, the device offers a promising alternative to conventional warm seed bathing or even chemical treatments by effectively inactivating fungi.

The primary objective of our research was to assess the performance of the UV-C device in disinfecting cabbage seeds. We aimed to determine the device's impact on seed quality in terms of reduction in Alternaria. By varying exposure time, wavelength, and intensity of UV-C radiation, we sought to identify optimal disinfection parameters that balance effective pathogen elimination.

To achieve accurate and reliable results, the testing procedures adhered to standardised protocols for all tests. We performed multiple replicates of each experiment to ensure robustness and consistency in the data. Moreover, we employed appropriate control groups and reference samples to establish a baseline for comparison and validate the efficacy of the UV-C device.

In the subsequent sections of this report, we will present a detailed description of our methodology, including the setup of the UV-C device and experimental procedures. We will then provide an analysis of the results obtained. Additionally, we will discuss the limitations encountered during the testing process and offer recommendations for further research and potential applications of the UV-C device in seed disinfection practices.

1.1. Problem definition

The disinfection of seeds is a critical process in the seed breeding industry to ensure optimal quality and therefore crop yield. However, traditional seed disinfection methods often rely on inefficient treatments. UV-C might be the next step in industry-leading disinfection methods.

In light of this objective, our research focuses on evaluating the efficacy of our custom-built UV-C device for seed disinfection. The primary objective addressed in this technical report is to determine the performance of this UV-C device in eliminating the fungus Alternaria on cabbage seed. By assessing the effectiveness of the UV-C device, we aim to address the following key questions:

- Can the UV-C device effectively inactivate (Alternaria) fungi on the seed?

- What are the optimal parameters, such as exposure time, the wavelength of UV-C, and the intensity of UV-C radiation for effective seed disinfection without compromising seed viability?
- What are the potential practical applications of the UV-C device in seed disinfection practices, and how does it contribute to sustainable and environmentally friendly agricultural practices?
- How does the UV-C device compare to the conventional warm water bathing method in terms of microbial reduction and overall seed quality?

The testing of Alternaria residue will be performed by Rijk Zwaan, located in De Lier, one of the world leaders in seed breeding. Rijk Zwaan will make use of PCR testing to determine the seed disinfection.

By providing answers to these questions, this technical report aims to contribute to the development and adoption of advanced seed disinfection technologies that offer efficient and more sustainable pathogen elimination while maintaining seed quality.

1.2. Hypothesis

Based on the literature, the hypothesis is that UVC-LED light can be used to disinfect the seeds. The Alternaria fungi can be eliminated by destroying the DNA and RNA structures by using a wavelength of 255 nm and the protein structures can be destroyed by using a combination of 275 nm and 285 nm.

2

Testing setup

Firstly the test setup will be explained. The test setup consists of a seed bed surrounded by two LED panels, which radiate UV-C light onto the seeds. A vibration motor will move and turn all seeds so that the seed surface is radiated as evenly as possible. Figure 2.1 provides an overview of the test setup.

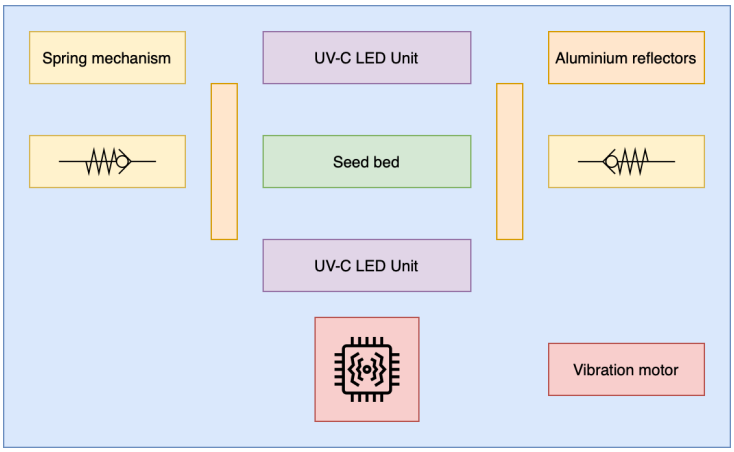


Figure 2.1: Seed distribution on a circular plate with 10cm diameter assuming 2.75 mm seed thickness.

The seedbed is a circular quartz plate with a diameter of 10 centimetres. The surface of the radiated area can then be calculated according to Equation 2.1.

$$A_{SeedBed} = \pi \cdot r^2 = \pi \cdot \left(\frac{d}{2}\right)^2 = \pi \cdot \left(\frac{10}{2}\right)^2 = 78.5 \text{ cm}^2 \tag{2.1}$$

Assuming a rapeseed diameter of 2 mm on average with an additional 0.75 mm margin per seed for stacking and placing seeds, the number of seeds that can be tested in one batch on the plate can be simulated. This can be observed in Figure 2.2.

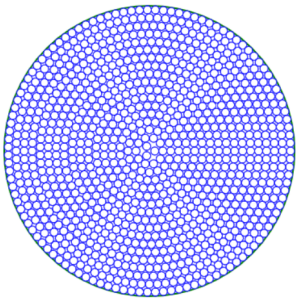


Figure 2.2: Seed distribution on a circular plate with 10cm diameter assuming 2.75 mm seed thickness.

The LEDs used in the setup and the specs are shown in Table 2.1. The optical output power is an essential parameter since the radiation time is dependent on this if a certain dose is desired to be radiated.

Wavelength	Typical Forward Voltage	Typical Forward Current	Dissipated Power	Optical Output Power
255	7.5V	100mA	0.75W	3.5mW
275	6.0V	85mA	0.51W	11.5mW
285	6.5V	90mA	0.585W	10mW
395	3.2V	75mA	0.24W	11.5mW

Table 2.1: Specifications of the LEDs used in the machine.

Radiation dose can be calculated using Equation 2.2. Each LED panel consists of 12 LEDs. Radiating is done from both sides, so the amount of LEDs and radiations surface is doubled.

$$Dose = \frac{P_{total} \cdot t}{2 \cdot A_{SeedBed}} = \frac{2 \cdot 12 \cdot P_{optical} \cdot t}{A_{SeedBed}} \tag{2.2}$$

Rewriting this equation to get the radiation time gives:

$$t = \frac{A_{SeedBed} \cdot Dose}{2 \cdot 12 \cdot P_{optical}} \tag{2.3}$$

3

Methodology

In order to achieve meaningful results, Rijk Zwaan has provided PCR testing to determine the residue of Alternaria and cabbage seed to disinfect. Unfortunately, time and test size constraints will limit the extensiveness of this research. Because of this, ten individual tests with different parameters can be performed, each of which will have a similar test as the control.

Before testing the actual cabbage seed, a pre-test is used to evaluate the radiation pattern of the UV-C LED disinfecting device. A silicon wafer is placed as a substitution for the quartz plate, which will act as a target. The silicon wafer, known for its light absorption characteristics, will be examined after exposure to UV-C. This pre-test aimed to visualize and analyze the distribution of UV-C radiation emitted by the device.'

Extending tests may be possible by the use of yeast. Irradiating yeast in its powder form commonly found in grocery stores can be an indicator of disinfection when added to sugar water. When the yeast is irradiated and killed, it should not create any bubbles in the sugar water.

3.1. Testing parameters for PCR tests

PCR tests can indicate the residue of genetic material, in this case Alternaria, on the seeds and can therefore be used to determine if the device works. Because of the limitations in test size of 10, some meaningful test setups with different parameters should be explored. Table 3.1 gives the chosen parameters for each individual test.
The following parameters can be set and researched; wavelength, motor speed, time and dose. The dose is related to the time and intensity of certain LEDs. It should be noted wavelength 3, $\lambda_3 95$, is not in the UV-C spectrum, but is UV-A. Literature suggested that pre-treatment with longer wavelengths of ultraviolet light can activate the pathogens and therefore make them weaker than in their "sleep".

3.1.1. Selecting proper dose

The dose is proportional to the exposure time and intensity of the LEDs. Choosing a useful exposure time is crucial for optimal test results. Based on literature, Alternaria can be eliminated by radiating with a dose of around $5kJm^{-2}$ [1, 2]. To investigate the relation of doses versus pathogen presence, seeds will be radiated by different radiation doses. The expected relation is a logarithmic curve. The doses which are chosen for different radiation levels are the following:

- $0.25\,\mathrm{kJ/m^2}$
- $1\,\mathrm{kJ/m^2}$
- $5\,\mathrm{kJ/m^2}$

According to Equation 2.3, the efficiencies from Table 2.1, and the area from Equation 2.1, the radiation time can now be calculated for each LED.

$$t_{\lambda_{255nm}(0.25kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 0.25 \cdot 10^{3}}{2 \cdot 12 \cdot 3.5 \cdot 10^{-3}} = 23.2s$$
$$t_{\lambda_{255nm}(1kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 1 \cdot 10^{3}}{2 \cdot 12 \cdot 3.5 \cdot 10^{-3}} = 92.9s$$
$$t_{\lambda_{255nm}(5kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 5 \cdot 10^{3}}{2 \cdot 12 \cdot 3.5 \cdot 10^{-3}} = 464.3s$$
$$t_{\lambda_{275nm}(0.25kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 0.25 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 7.07s$$
$$t_{\lambda_{275nm}(1kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 1 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 28.3s$$
$$t_{\lambda_{275nm}(5kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 5 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 142.2s$$
$$t_{\lambda_{395nm}(0.25kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 0.25 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 7.07s$$
$$t_{\lambda_{395nm}(1kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 1 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 28.3s$$
$$t_{\lambda_{395nm}(5kJm^{-2})} = \frac{78.5 \cdot 10^{-4} \cdot 5 \cdot 10^{3}}{2 \cdot 12 \cdot 11.5 \cdot 10^{-3}} = 142.2s$$

3.1.2. Parameter selection

Test plan	Motor	Hot air [°C]	λ_{255nm}	λ_{275nm}	λ_{395nm}	time [s]	Dose [kJ/m^2]
Test 1, control	off	-	-	-	-	-	-
Test 2	off	-	100%	-	-	92.9	1
Test 3	on	40	100%	-	-	92.9	1
Test 4	on	-	100%	-	-	23.2	0.25
Test 5	on	-	100%	-	-	464.3	5
Test 6	on	-	100%	-	-	92.9	1
Test 7	on	-	-	100%	-	7.07	0.25
Test 8	on	-	-	100%	-	28.3	1
Test 9	on	-	-	100%	-	142.2	5
Test 10	on	-	100%	-	100%	92.9 λ_{255nm}, 28.3 λ_{395nm}	1+1

Table 3.1: Tested parameters

The tests setups in table 3.1 explained:

- Test 1 will be a control without UV treatment, motor vibration and hot air pre-treatment.
- Test 2 tests the effectiveness of the motor.
- Test 3 tests the effectiveness of a hot air pre-treatment. By comparing tests 1, 2, 3 and 6 the effectiveness of the motor and the hot air-pre-treatment can be determined.
- Test 4 tests the effectiveness of λ_{255} with a dose of 0.25 [kJ/m^2].
- Test 5 tests the effectiveness of λ_{255} with a dose of 5 [kJ/m^2].
- Test 6 tests the effectiveness of λ_{255} with a dose of 1 [kJ/m^2].
- Test 7 tests the effectiveness of λ_{275} with a dose of 0.25 [kJ/m^2].
- Test 8 tests the effectiveness of λ_{275} with a dose of 1 [kJ/m^2].
- Test 9 tests the effectiveness of λ_{275} with a dose of 5 [kJ/m^2].
- Test 10 tests the effectiveness of pre-treatment with a longer wavelength, in UV-A, λ_{395} in combination with λ_{255}. Each wavelength will deliver a dose of 1 [kJ/m^2], making the total 2 [kJ/m^2]. To be compared with test 6.

Please note that test 10 actually delivers a total dose of 2 [kJ/m^2]. This is not expected to spoil the result because UV-A is not known for strong germicidal and disinfecting capabilities.

3.2. Testing parameters for silicon tests

3.2.1. Selecting proper dose

Silicon wafers are photoreactive for wavelengths between $300\,\text{nm}$ and $500\,\text{nm}$. The silicon wafers can give an indication about the irradiation pattern and uniformity of radiation on the plate.

The proper dosage of UV-C radiation in which results would be visible is about $40\,\text{mJ/cm}^2$. As stated earlier, the LEDs have an optical power output of:

- 3.5 mW for a 255 nm LED
- 11.5 mW for a 275 nm LED
- 10 mW for a 285 nm LED
- 11.5 mW for a 395 nm LED

As stated earlier, the 285 nm LEDs don't work properly and therefore cannot be tested. the area of the plate is 78.5 cm^2. There are 12 LEDs for each LED type. As the wafer is only photo reactive from the top, only the top PCB will be turned and used for irradiation. Note silicon is mostly reactive in the 300-500 nm region, so the 395 nm LEDs are expected to yield the most promising results.

$$P_{area,255nm} = \frac{3.5 \cdot 12}{78.5} = 0.53\,\text{mW/cm}^2$$

$$P_{area,275nm} = \frac{11.5 \cdot 12}{78.5} = 1.75\,\text{mW/cm}^2$$

$$P_{area,395nm} = \frac{11.5 \cdot 12}{78.5} = 1.75\,\text{mW/cm}^2$$

Meaning, for $40\,\text{mJ/cm}^2$ the device should be turned on for a total of 74.8 seconds for the 255 nm LEDs and 22.8 seconds for the 395 and 275 nm LEDs. Some meaningful parameter testing is given in Table 3.2.

Parameter selection					
Test plan	Reflector	Scanner	λ_{395nm}	time [s]	Dose $[mJ/cm^2]$
Test 1	no	no	100%	22.8	40
Test 2	no	no	100%	45.6	80
Test 3	yes	no	100%	22.8	40
Test 4	yes	no	100%	45.6	80
Test 5	no	yes	100%	22.8	40
Test 6	yes	yes	100%	45.6	80
Test 7	yes	yes	100%	22.8	40
Test 8	no	yes	100%	45.6	80

Table 3.2: Tested parameters

The following tests are conducted:

- Test 1 tests the radiation pattern of the 395 nm LEDs at the suggested dose.
- Test 2 tests the radiation pattern of the 395 nm LEDs at double the suggested dose.
- Test 3 and 4 tests the same parameters with the reflector put onto the sides of the seed bed.
- Test 5 till 8 tests the same parameters again so that the test set is doubled and on of the sets can be used to directly see results after scanning it.

3.3. Testing yeast

To get an indication of the radiation needed to destroy fungi and to verify the calculations and effectiveness of the tests above the machine can be tested by loading yeast into the seed bed. To test the viability of yeast a sugar solution in water can be used. Testing this requires a similar concentration of

sugar to test normal yeast and yeast radiated with UV-C light. Testing was done by filling up a translucent glass of water with 200 mL of water and mixing 30 grams of sugar. One of the glasses can be used as a control test by monitoring the carbon dioxide (CO_2) production. This can be compared with the production of carbon dioxide from the yeast radiated with the light in the machine. Radiation is done by a dose of $0.5\,\mathrm{kJ/m^2}$.

3.4. Testing protocol

To ensure accuracy and reliability in the evaluation of the UV-C device's performance for seed disinfection, a standardized testing procedure was implemented. The procedure aimed to minimize potential errors and inconsistencies, thereby providing consistent, reliable, and comparable results throughout the testing process.

First, a controlled testing environment was established. This environment maintained similar conditions as the tested seeds in terms of temperature, humidity, and lighting conditions to eliminate any external factors that could influence the test outcomes.

The seed-disinfection device created by our team was employed for each trial to ensure reproducibility. The seeds used in the tests were sourced from the same batch and were of uniform size and quality. To guarantee consistency, a specific number of seeds were selected for each trial, namely 500, and they have been evenly distributed across the test area.

The distance between the UV-C device and the seeds was kept constant throughout the testing process. This distance and LED placement were determined based on prior research and optimization.

To further minimize errors, the exposure time was standardized for each test. Two predetermined durations were chosen based on literature studies for effective seed disinfection of fungi (Alternaria). This ensured that the seeds received the same UV-C dosage during each trial, allowing for an accurate comparison of the disinfection outcomes.

In between tests, the quartz plate, which holds the seed, is thoroughly disinfected by radiation without seeds in order to assure no pathogens are carried over to the next test batches.

Finally, when placing and removing the seeds from the quartz testing plate, no physical contact with hands or non-sterile equipment is permitted. The seeds are placed in sterilized bags outside the device in order to assure safe and sterile transport to Rijk Zwaan.

4

Results

4.1. PCR test results

The test results following from the PCR tests performed by Rijk Zwaan are given in Table 4.1. The subsequent graphs are given in Figures 4.1, 4.2, and 4.3.

	Sample	FAM (A.brassicicola)	VIC (A.brassicicola)	Cy5 (A. brassicae)	TxR (IAC Acat)	Uitslag
S	1.1	20,98	24,80	N/A	19,48	A.brassicicola
	2.1	21,00	24,63	37,74	19,36	A.brassicicola
	3.1	20,21	24,04	N/A	19,49	A.brassicicola
	4.1	21,31	25,14	N/A	19,40	A.brassicicola
	5.1	21,78	25,40	N/A	19,46	A.brassicicola
	6.1	21,14	25,04	N/A	19,49	A.brassicicola
	7.1	20,88	24,48	N/A	19,67	A.brassicicola
	8.1	21,17	25,16	N/A	20,41	A.brassicicola
	9.1	20,70	24,54	N/A	19,62	A.brassicicola
	10.1	21,58	25,42	N/A	19,47	A.brassicicola
	1.2	21,10	25,06	N/A	19,55	A.brassicicola
	2.2	20,75	24,43	38,81	19,68	A.brassicicola
	3.2	21,37	25,27	36,79	19,81	A.brassicicola
	4.2	21,11	24,79	N/A	19,69	A.brassicicola
	5.2	21,81	25,55	N/A	19,76	A.brassicicola
	6.2	22,67	26,38	N/A	20,62	A.brassicicola
	7.2	21,05	24,82	N/A	19,75	A.brassicicola
	8.2	21,05	24,86	N/A	19,45	A.brassicicola
	9.2	21,20	25,04	N/A	19,49	A.brassicicola
	10.2	21,12	25,11	N/A	19,72	A.brassicicola
C	QC_NTC	N/A	N/A	N/A	N/A	Oke
	QC_NPC	N/A	N/A	N/A	21,57	Oke
	QC_NAC	N/A	N/A	N/A	23,20	Oke
	QC_PC	19,69	18,03	19,39	N/A	Oke

Table 4.1: PCR test results from Rijk Zwaan. The S indicates the samples, the C indicates the PCR-control tests.

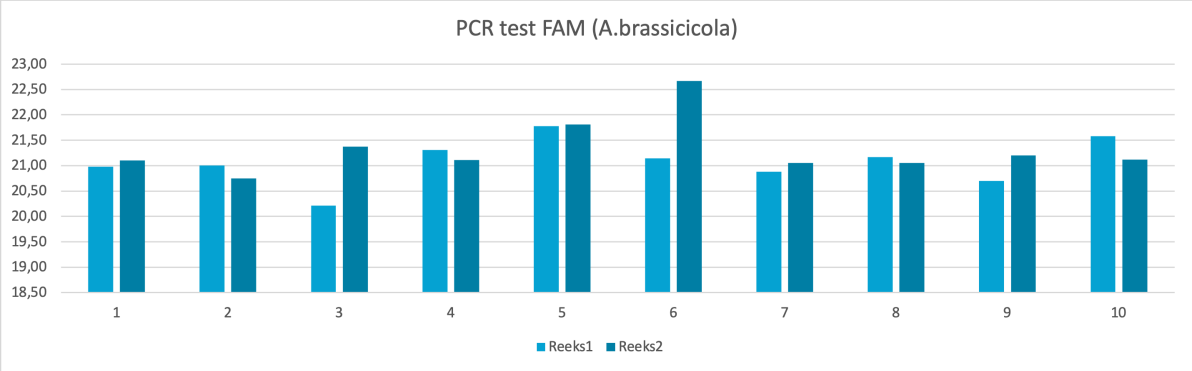


Figure 4.1: The PCR Test tested with FAM.

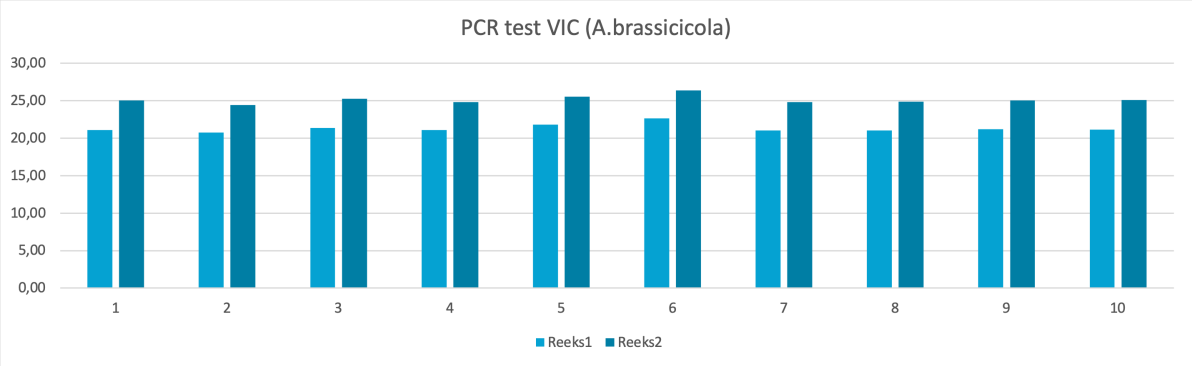


Figure 4.2: The PCR Test tested with VIC.

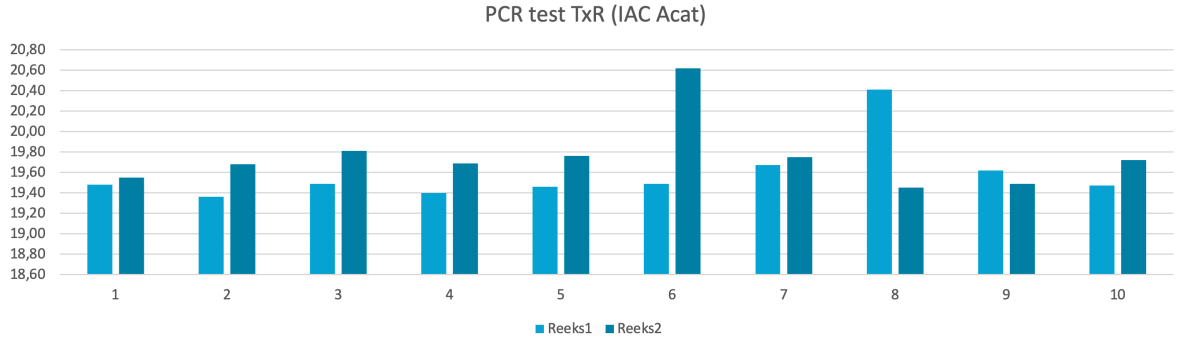
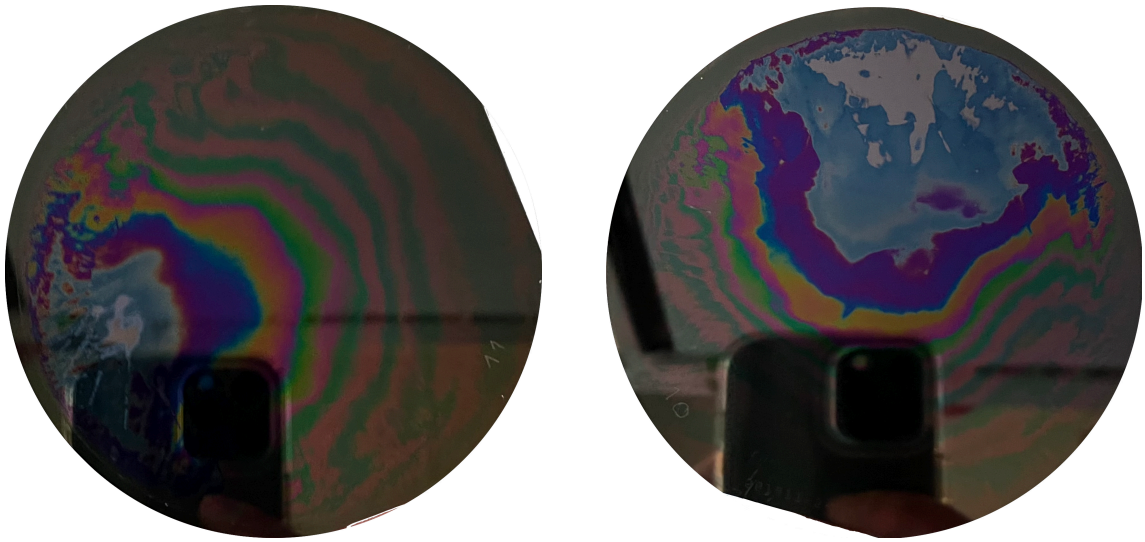


Figure 4.3: The PCR Test tested with TxR.

4.2. Silicon wafer test results

The results for the silicon wafer test will mostly be visual comparison and therefore the data is qualitative. The results are given in Figure 4.4.



(a) The silicon wafer radiated with a dose of $0.4\,\mathrm{kJ/m^2}$.

(b) The silicon wafer radiated with a dose of $0.8\,\mathrm{kJ/m^2}$.

Figure 4.4: Silicon wafer radiated with UV-C light.

The silicon wafers are developed in a basic solution so that the exposed parts are dissolved and the radiation pattern becomes visible.

4.3. Yeast test results

Yeast was tested in two glasses of water. Results are that the yeast from the control test was actively producing CO_2 and the radiated yeast was notably less active. The foam head as a result of the production of carbon dioxide was way thinner than the control test.

5 Discussion

It is important to note that even with strict adherence to standardised procedures, it is possible for some degree of variability to exist due to biological variations among seeds but also slight human errors. However, by implementing controls and consistent testing practices, the impact of such variations can be minimised, ensuring reliable and valuable data for the evaluation of the UV-C device's performance in seed disinfection.

Also, the test results of the PCR tests measure how much DNA or RNA there is still left for a certain test string after the tests have been executed. In the case of these test results, test strings of the pathogen A. brassicicola were used as a reference. From this, it could be investigated how much of these pathogen material is still left in the samples. However, the PCR results do not point out whether these structures are viable or not. Seeds can be disinfected while still detecting their structures. If those structures are not viable anymore, they are not harmful to the plants, which is the desired result. To summarize, the PCR method is not a good method to conclude whether seeds are disinfected, but only to detect if DNA or RNA structures are still present.

Actually, this was not yet known at the beginning of the project, but this is a possible declaration for the test results, which do not show any significant results. For future work, the ELIZA method is probably a better method to test the effectiveness of the machine.

This method does not test the DNA or RNA structures, but it tests the protein structures. Detecting proteins with enzymes is a much more specific process, so the viability of the pathogens can be determined more reliably.

Another method which would probably work better than the PCR method is the blotter or malt agar method used by the International Seed Health Association [4]. In short how it works: treated seeds get placed on a petridish on a medium, filter/blotter paper or malt agar. After, the seeds get incubated and then freezed. In the end, the seeds germinate and Alternaria grows too if it is present on the seeds. This takes a bit more time than the PCR method, however, less time than planting the seeds and examining the leaves.

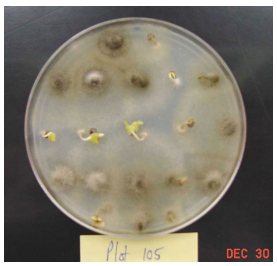


Figure 5.1: Cabbage seed plated onto a semi-selective agar medium to detect Alternaria brassicicola [3]

The results of the silicon wafer test can be used to verify the dose of the LEDs. The radiation pattern cannot be determined reliably because the wafers are developed by hand which is much less accurate than doing it automatically by a machine.

6

Conclusion

From the results in chapter 4, it can be seen that there are no significant differences in the results. Although it cannot be concluded that the machine can successfully disinfect seeds, the PCR method is not the best method to verify this. As explained in the Discussion section, future work is needed to show if the machine can disinfect seeds. This conclusion is colourable since the yeast test results show that UV-C light can eliminate fungi. The integration of the machine is successful. All submodules are tested individually to ensure reliability and safety for both the electronic and user side. The integration is also tested and the machine can select the user input and set the settings into the machine for reliable testing. The verification of the dose of the LEDs is done by studying the colour pattern. It turns out the dose is correctly radiated onto the seed bed. The machine has the potential to show the effectiveness of the UV-C light and future work can be interesting to execute.

References

[1] Nan Jiang et al. “Effects of ultraviolet-c treatment on growth and mycotoxin production by Alternaria strains isolated from tomato fruits”. In: *International Journal of Food Microbiology* 311 (2019), p. 108333. ISSN: 0168-1605. DOI: https://doi.org/10.1016/j.ijfoodmicro.2019.108333. URL: https://www.sciencedirect.com/science/article/pii/S0168160519302636.

[2] Dongqi Guo, Lixia Zhu, and Xujie Hou. “Combination of UV-C Treatment and Metschnikowia pulcherrimas for Controlling Alternaria Rot in Postharvest Winter Jujube Fruit”. In: *Journal of Food Science* 80.1 (2015), pp. M137–M141. DOI: https://doi.org/10.1111/1750-3841.12724. eprint: https://ift.onlinelibrary.wiley.com/doi/pdf/10.1111/1750-3841.12724. URL: https://ift.onlinelibrary.wiley.com/doi/abs/10.1111/1750-3841.12724.

[3] Lindsey du Toit. *Cabbage seed plated onto a semi-selective agar medium to detect Alternaria brassicicola.* https://mtvernon.wsu.edu/path_team/cabbage.htm#alternariablackspot. 2023.

[4] International Seed Health Association. “International Rules for Seed Testing 2022: Validated Seed Health Testing Methods”. In: 2023.Number 1 (2023), i-7–6(6).