Combined UVC and Plasma Application for Seed Disinfection

The UVC Unit

Miryam Majdoubi & Marit Woudstra

Delft University of Technology

TU Delft

Combined UVC and Plasma Application for Seed Disinfection

The UVC Unit

by

Miryam Majdoubi & Marit Woudstra

Thesis committee:	Prof. dr. ir. Jan van Turnhout	Supervisor
	Dr. Luutzen Wymenga	Interim Supervisor
	Dr. Ilke Ercan	Committee Chair
Project Duration:	April, 2026 - June, 2026	
Faculty:	Faculty of Electrical Engineering, Delft	

Cover: by JJ Ying

TU Delft

Abstract

Seed-borne pathogens present a significant threat to agricultural productivity, with severe infestations potentially leading to food shortages. Although chemical pesticides remain the predominant method for seed disinfection globally, their environmental impact necessitates the exploration of sustainable alternatives. This project presents the development and evaluation of an eco-friendly ultraviolet-C (UVC) irradiation system designed to provide targeted, uniform germicidal radiation at a peak wavelength of 265 nm. The system was tested on cabbage seeds infected with Xanthomonas campestris pv. campestris and Alternaria brassicicola using a PWM-driven circuit to apply various duty cycles. It was also designed to integrate with a plasma unit and high-voltage supply developed by collaborating subgroups. Experimental findings indicate that UV-C treatment alone can suppress pathogen growth, with duty cycles of 30% and 50% occasionally delivering germicidal efficacy comparable to continuous-wave exposures.

Contents

1 Introduction **1**
- 1.1 State-of-the-art analysis . . . 1
- 1.2 Problem Definition . . . 1
- 1.3 Situation Assessment . . . 2
- 1.4 Scoping and Bounding . . . 2
- 1.5 Group-division . . . 2
- 1.6 Report Structure . . . 3

2 Chapter 2: Theoretical Background **4**
- 2.1 The electromagnetic spectrum . . . 4
- 2.2 Mechanism of action . . . 5
  - 2.2.1 Peak absorption . . . 6
- 2.3 Pathogens and Dosage . . . 7
- 2.4 Continuous and Pulsed UV Exposure . . . 7
- 2.5 Dormancy . . . 7

3 Programme of Requirements **9**
- 3.1 Requirements . . . 9
  - 3.1.1 Mandatory Requirements . . . 9
  - 3.1.2 Trade-Off Requirements . . . 10

4 Design and Simulations **11**
- 4.1 Design . . . 11
  - 4.1.1 Subunits . . . 11
  - 4.1.2 Chosen Materials . . . 11
- 4.2 Simulations . . . 12
  - 4.2.1 PWM Circuit . . . 12
  - 4.2.2 LED Irradiation . . . 15

5 Hardware and Firmware **18**
- 5.1 PCB Design . . . 18
  - 5.1.1 LED Panel . . . 18
- 5.2 Coding . . . 19
  - 5.2.1 PWM . . . 19
  - 5.2.2 Sensors . . . 19
  - 5.2.3 Display . . . 20

6 Testing **21**
- 6.1 Individual Subsystem Testing . . . 21
  - 6.1.1 Pre-Testing . . . 21
  - 6.1.2 Testing Parameters . . . 22
  - 6.1.3 UVC disinfection . . . 22
- 6.2 Integration . . . 23
  - 6.2.1 Subunits . . . 23
  - 6.2.2 Plasma and HV source . . . 23

7 Results **24**
- 7.1 The Banana Test . . . 24
- 7.2 LTSpice Verification . . . 24
- 7.3 PCR and agar tests . . . 25
  - 7.3.1 UVC solo . . . 26

7.3.2 UVC and Plasma . . . 27
7.4 Seed Germination Test . . . 30

8 Discussion **31**
8.1 General . . . 31
8.2 UVC alone . . . 31
8.3 Integrated Plasma and UVC . . . 32
8.4 Future Work . . . 32

9 Conclusion **33**

References **34**

A Ethical Considerations **38**
A.1 Societal impacts . . . 38
A.2 Ethical aspects of development . . . 38
A.3 Ethical evaluation . . . 38
A.4 Reflection on design choices . . . 39
A.5 Forward-looking responsibility . . . 39

B Source Code **40**
B.1 Python Code for LED Configuration . . . 40
B.2 C Code Microcontroller . . . 44
B.2.1 PWM . . . 45
B.2.2 Rotary Encoder . . . 52
B.2.3 Ozone Sensor . . . 56
B.2.4 Display . . . 60
B.2.5 UV Sensor . . . 64
B.2.6 Temperature and Humidity Sensor . . . 64

C Other Figures **65**
C.1 LTSpice . . . 65
C.2 Schematics . . . 67
C.2.1 Schematic of LED Panel . . . 67
C.3 PCB Designs . . . 68
C.3.1 PCB Design of LED Panel . . . 68
C.4 Test set-up . . . 68
C.5 Results . . . 70
C.5.1 LTSpice Verification Measurements . . . 70
C.5.2 Agar Tests . . . 71
C.5.3 Germination Test . . . 74

D Tables **75**
D.1 PCR Results . . . 75
D.1.1 UV . . . 75
D.1.2 UV and Plasma . . . 76
D.2 Agar Results . . . 77
D.2.1 UV . . . 77
D.2.2 Intermediate . . . 77
D.2.3 Final . . . 77
D.2.4 UV and Plasma . . . 77

E Calculations **78**
E.1 PWM Circuit . . . 78
E.1.1 LEDs . . . 78
E.1.2 Average LED Forward Voltage . . . 78

1 Introduction

To graduate for the Bachelor's degree in Electrical Engineering, a final Bachelor Thesis Project (BAP) must be completed to showcase the knowledge and engineering skills gained over the course of the studies has to be carried out. Working in groups, students are tasked with a certain problem and together the whole team should find an engineering solution by doing research, building prototypes, testing and improving the design. The final product, together with the results are presented at the end of the BAP period.

1.1. State-of-the-art analysis

With the global population increasing each day, to an estimated 10 billion people in 2050, there will have to be drastic changes made in the production of food. To sustain all these people the food production worldwide will have to increase by around 60% [1][2][3]. By minimising agricultural food waste in the production of crops, a big part of this problem can be resolved. Right now part of the crops are infected by bacteria, fungi or other pathogens, which cause the crops not to grow properly or even die. These infections affect all sorts of crops and can cause losses of 17% to about 30% [1], depending on the type of crop used.

To increase the yield of crops and battle these infections, pesticides are used in large quantities. It depends highly on the type of crop and where in the world it is planted, but over 70%, and in many cases even more than 90%, of crops still get disinfected with pesticides [4] [5]. This amounts to about 2.7 billion kilograms of pesticides used yearly [6]. In the last 40 years the usage has increased by 50%.[6] Alternative ways to disinfect without using strong chemicals have been found, such as using warm water baths, but these still produce a lot of waste.

Fortunately, another growing market is that of using plasma and UV to disinfect, the growth for plasma is around 15%[7], and UVC is one of the fastest growing applications for decontamination [8]. Which is a good progress, since the regulations around the use of hazardous chemicals is being regulated more. While standalone UV-C irradiation and cold atmospheric plasma treatments have individually proven viable, their combined effectiveness is yet to be discovered. Both technologies have demonstrated strong efficacy in standalone configurations across various sterilization domains. Ultraviolet-C radiation, particularly within the germicidal spectrum of 250 to 280 nanometers, works by penetrating microbial cell walls where it is directly absorbed by cellular DNA and RNA. Concurrently, cold atmospheric plasma operates by ionizing ambient gases, creating charged particles, and reactive oxygen and nitrogen species (RONS) such as ozone. These reactive species interact directly with biological membranes, inducing severe oxidative stress, thus killing the pathogens.

1.2. Problem Definition

The core challenge addressed in this project is verifying the integration of several eco-friendly seed disinfection methods. While chemical pesticide application is a simple approach to the problem of disinfection, finding a method without using products that have an environmental burden on the planet

and is as effective is still a challenge. Specifically when high heat can damage the seeds, high humidity or not a good enough drying process can cause premature germination.

1.3. Situation Assessment

Agricultural disinfection has historically relied heavily on chemical treatments (amounting to 2.7 billion kilograms of pesticides globally)[6] and hot water baths. These methods are either using highly dangerous substances to the human body or produce large quantities of waste to the environment. On top of that, the use of pesticides is slowly being restricted, while there is a growing demand for high-quality seeds.

Seed disinfection methods can be split up into three categories: chemical, biological and physical. Currently companies like Bejo are actively investing and performing research into disinfection methods that do not rely on traditional chemical pesticides. Procedures like steam-vacuums, where the seeds are placed into a vacuum chamber and then treated with superheated steam or humid air, or a process called thermal-rinse sequence, where the seeds for a short period of time are exposed to an intense heat which only kills the bacteria on the seeds, since they are single-celled organisms which can not handle as much as the seeds. Furthermore, current alternative chemical disinfection methods look into utilizing substances such as hydrogen peroxide, which cleans the outside of the seed through oxidation before decomposing into water and oxygen molecules. Lastly, biological disinfection methods are being used as a way to suppress pathogens using natural micro-ecosystems. All of these options demonstrate that the industry is slowly shifting toward non-chemical alternatives.

For an ideal future, chemical pesticides will not be in use anymore in the agricultural sector, thus reduce the toxic effect they have on humans. As for hot water baths, they are no longer in use, so there will not be an excessive amount of water waste. Instead, methods like plasma and UVC radiation create a successful way of fighting pathogens and increase crop production, without producing any waste.

1.4. Scoping and Bounding

The scope of this project will be designing, building and testing a system that can disinfect seeds by combining UVC and cold-plasma. The final product will be a validation to see whether using a combination of both techniques results in better seed disinfection, than using either individual technique. On a smaller level the UVC design will be tested on whether using continuous versus pulsed UVC is more effective.

To achieve a functional UV system, an STM32 microcontroller is utilised, which will control a system consisting of UVC LEDs, a PWM circuit, several sensors and a display.

1.5. Group-division

In figure 1.1, the operational overview of the integrated disinfection set-up can be seen. It consists of a tray containing the Brassica (cabbage) seeds, which is irradiated by the cold plasma unit directly above it. This plasma unit is designed by another subgroup in this project. A third group designed a high-voltage power supply that can power the plasma panel. At $3\ cm$ above the plasma panel, the UVC panel is positioned. This will be controlled by a microcontroller, and sensor values will be read out by a display. This report will handle the design and testing of this part of the set-up.

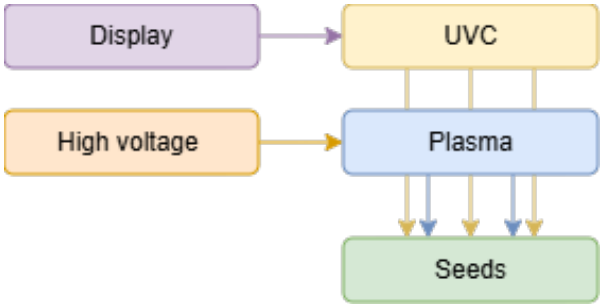


Figure 1.1: System Overview

1.6. Report Structure

The remainder of this report details the engineering of the UVC integration sub-system. Chapter 2 explains the theoretical background of UVC radiation, and its germicidal effect and the idea of dormancy of seeds. Chapter 3 establishes the Programme of Requirements (PoR) and system constraints. Chapter 4 presents the initial design choices and simulations for both the circuitry and the LED placements, performed in LTSpice and Python, respectively. Chapter 5 outlines the physical design and testing setup, as well as coding required for controlling the system. Chapters 6 and 7 present the testing of the individual and integrated set-up, as well as the results of performed tests. Finally, Chapters 8 and 9 give the discussion on this project as well as the future work that can be done, followed by the conclusion of this thesis.

2

Chapter 2: Theoretical Background

2.1. The electromagnetic spectrum

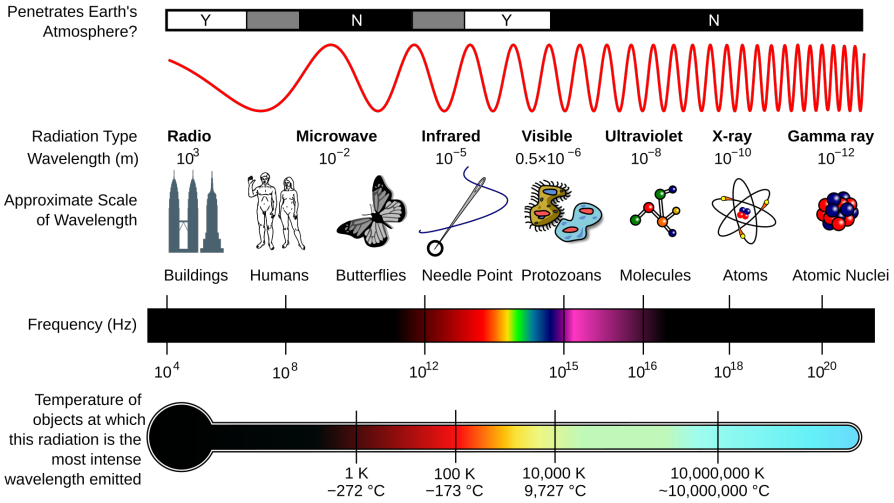


Figure 2.1: The full electromagnetic spectrum. Reproduced from [9].

The electromagnetic (EM) spectrum describes all electromagnetic radiation ordered by wavelength (λ) or frequency (f). Visible light makes up only a small part of the electromagnetic spectrum. The range of wavelengths that can be seen by the human eye (380 - 780 nm) is only a small part of it. Looking at wavelengths that are even smaller, we arrive in the ultra-violet (UV) part of the spectrum (100-380 nm) [10] [11] [12]. The photons in this part of the spectrum carry more energy because the wavelengths are smaller [13]. The formula that describes this phenomenon using the Planck constant (h) and the speed of light (c) is:

$$E = \frac{hc}{\lambda} \tag{2.1}$$

The UV band can itself be divided into smaller bands in multiple ways. A common classification is long-wave UV-A, medium-wave UV-B, and short-wave UV-C [14]. UV-A emits between 315 to 400 nm, and its wavelength is the largest relative to the other UV bands. UV-B emits shorter wavelengths, between 280 and 315 nm. UV-C, Sometimes described as ”germicidal UV”, emits between 200 to 280 nm. Lastly, there is vacuum-UV with the smallest wavelengths, between 100 and 200 nm [14]. Occasionally, vacuum-UV is merged with UV-C so that UV-C is said to run from 100 to 280 nm [10].

As wavelengths decrease within the UV spectrum, photon energy increases according to $E = \frac{hc}{\lambda}$. For many common materials, these higher-energy photons approach or exceed energies associated with electronic transitions, resulting in an increase in absorption. This generally results in less transmission through many common materials for the shorter wavelengths compared to the longer wavelengths in general, although this is entirely dependent on the material and is not a hard rule [15] [16].

UV-A radiation from the sun passes through the atmosphere with relative ease, and makes up 95% of the UV that reaches the earth, with the other 5% being UV-B. By contrast, UV-C essentially doesn't reach Earth's surface as it reacts with the oxygen (O_2) and ozone (O_3) [17]. Most common UV protection measures (e.g. sunglasses and sunscreen) are therefore designed and tested to protect against UV-A and UV-B. It is therefore important to ensure safety measures that protect against UV-C exposure.

2.2. Mechanism of action

UV-C is also called germicidal UV because of its capability of inactivating all sorts of microbes (e.g. bacteria, fungi, and viruses) [18]. The mechanisms involve interactions between the UV-C photons and the genetic material and proteins of the pathogen it reaches [19].

All known living organisms carry their genetic material in polymers called ribonucleic acid (RNA) and deoxyribonucleic acid (DNA) [20] [21]. While DNA most often contains the genetic code, different types of RNA are used to convert those blueprints into the actual genetic material. They resemble each other closely in structure, both containing a sugar-phosphate backbone with nitrogenous bases attached. The unit that makes up these polymers is called a nucleotide and this unit contains a single ribose sugar, one of four possible nitrogenous bases, and phosphate group. In the case of DNA, the sugar is deoxidized, there are often two complimentary strands instead of single strands, and one of the possible four bases is thymine rather than uracil as is the case for RNA. The nitrogenous bases can be divided into two groups: pyrimidines which have a single carbon ring, and purines which have two rings. Cytosine, thymine (exclusive to DNA), and uracil (exclusive to RNA) are pyrimidines, while adenine and guanine are purines [22].

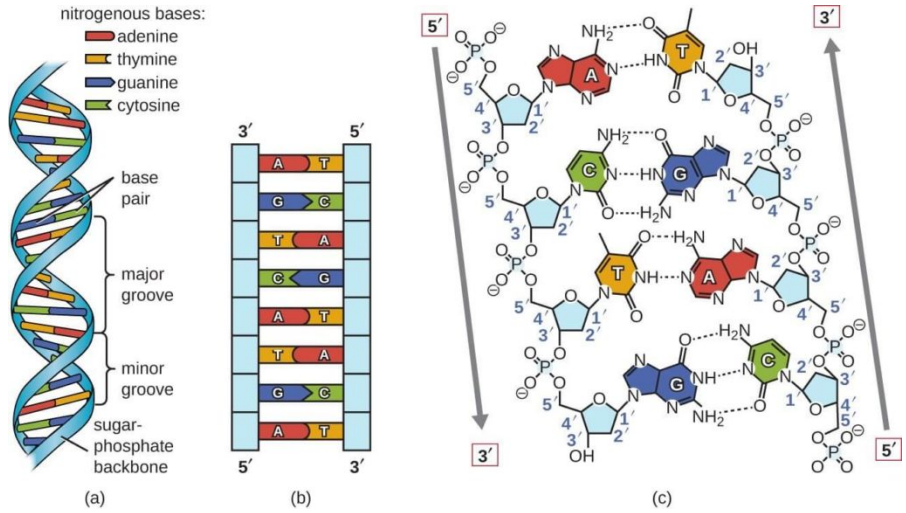


Figure 2.2: The structure of DNA. Reproduced from [23].

When a UV photon interacts with DNA it can excite the electrons in the carbon double bonds found in the rings of two neighboring pyrimidines into forming bonds with each other, creating dimerized pyrimidines [24] [19]. A similar phenomenon occurs when UV light hits RNA, while for proteins the primary damage happens through various different pathways [25].

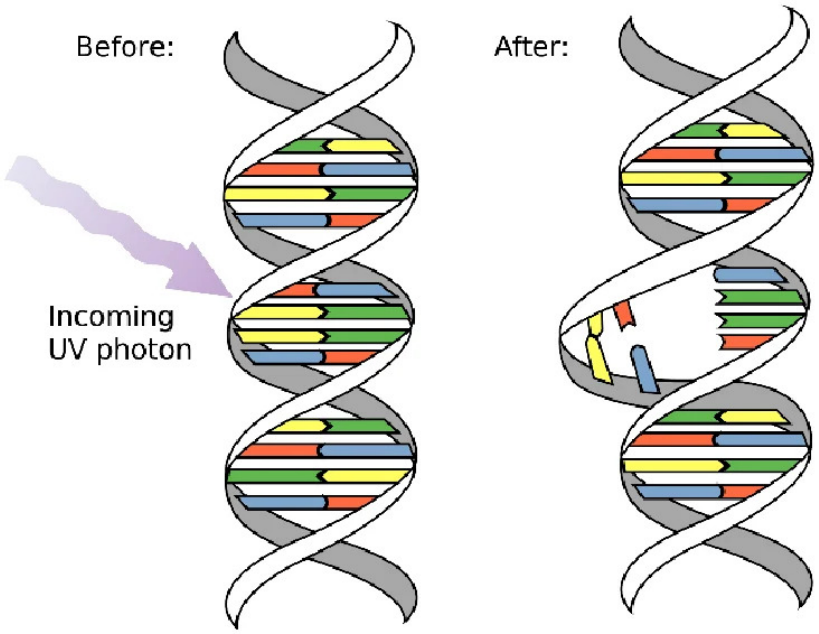


Figure 2.3: UV light hitting nucleic acids and forming dimerized pyrimidines. Reproduced from [26].

2.2.1. Peak absorption

To maximize the efficiency of UV treatment, it is important to select those wavelengths that maximize the absorption of UV. When considering various pathogens, this can be done by selecting for wavelengths that are maximally absorbed by nucleic acids (DNA, RNA) and proteins to incur damage in the genetic material of the target, as well as the organic material that performs tasks essential for growth, repair, and multiplication. Looking at Figure 2.4, the peak absorption wavelength for nucleic acids is around 260 nm while the peak wavelength for proteins is around 280 nm as well as 220 nm (not shown in figure) [27].

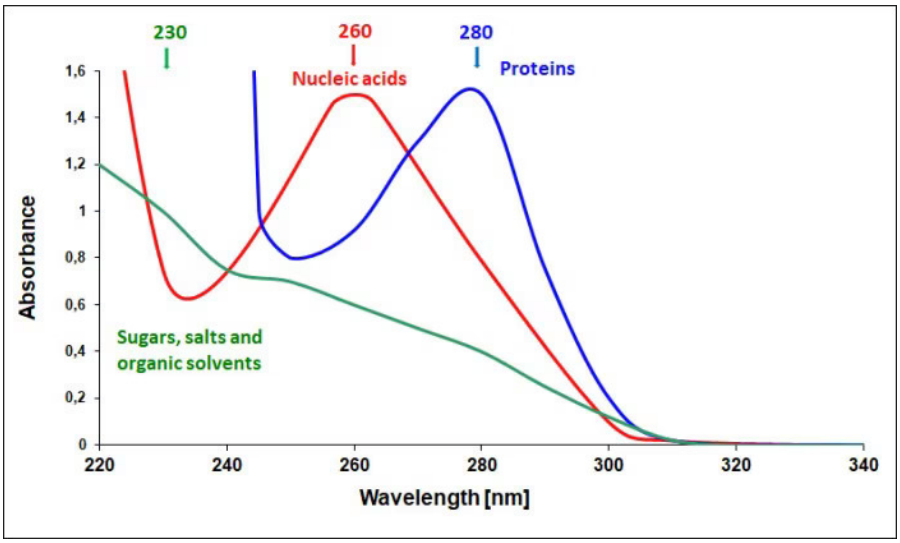


Figure 2.4: Peak absorption wavelengths of differing organic material. Reproduced from [28].

2.3. Pathogens and Dosage

The dosage of accumulated UV is measured in irradiance, often mJ/cm^2. This is because it represents an accumulated amount of light over time, usually in the order of millijoules:

$$Irradiance(mW/cm^2) \times Duration(seconds) = Energy\ Density(mJ/cm^2) \tag{2.2}$$

Different pathogens have been found to require different dosages of the same wavelength of UV-C radiation (most often 254 nm) to achieve an inactivation rate over 99%. Most of the microbes that have been extensively tested on are the ones that infect humans rather than crops. The pathogens of interest for this research are the bacterium Xanthomonas campestris pv. campestris (Xcc) which can cause black rot in cabbages, and the fungus Alternaria brassicicola (Abco) which can cause black spot disease in cabbages [29][30]. To get an indication of the dosage that would be required to inactivate these pathogens it can be helpful to look at the approximate range that is required to inactivate other bacteria and fungi at 254 nm [31] [32]. It can be seen that for most bacteria a dosage between 1 - 15 mJ/cm^2 suffices, and for most fungi the range is between 6 - 40 mJ/cm^2. Additionally there are also some more resistant bacteria with required dosages up to 95.0 mJ/cm^2 and fungi that require dosages up to 226.0 mJ/cm^2.
However, dosage alone does not fully determine disinfection efficacy. A pathogen may receive 0.01 mW/cm^2 for an hour, or 0.6 mW/cm^2 for a minute, to get a dose of 36 mJ/cm^2. Because the targets of the dosage are capable of repair and multiplication, the time-frame wherein this occurs is also relevant [33].

2.4. Continuous and Pulsed UV Exposure

This leads to the question of what the effects of using pulsed exposure to UV might have on different pathogens. What the effects are of keeping the intensity constant and varying the duty cycle and thus the total dosage. Or keeping the total dosage constant but varying the time based on the duty cycle. Current research shows varying results with likely differences between pathogens. In certain cases, a lower duty cycle was deemed more effective than higher duty cycles, while in others it performed worse [34][35]. At other times, they might be equally effective [36][37].
Even when pulsed UV only performs similarly to continuous UV, it offers the benefit of being more energy efficient when the irradiation time between the two is kept constant. And the lower the duty cycle, the more energy efficient it can be. Another potential benefit that can accompany the use of pulsed UV, is the ability to use cheaper UV-LEDs at higher intensities because some LEDs have a higher rated pulsed current than continuous current.

2.5. Dormancy

Bacteria and fungi have the ability to enter a protective, highly resistant, dormant state to survive extreme conditions under high stress [38][39]. This can make them harder to detect on top of being capable of surviving harsh environments [40] [41]. When these pathogens survive past the disinfection stage, they are once again capable of re-entering a normal state where they can multiply and harm their host. It is therefore of interest to discover how to keep as many of the targeted pathogens as possible out of this dormant state before and during the disinfection stage.

Research on this topic is ongoing. Bacterial spores possess receptors that act like nutrient-gated ion channels that can trigger awakening from dormancy [42]. This suggests that addition of specific nutrients to the environment may promote germination and resuscitation of dormant bacteria. Not a lot is known on the mechanisms for resuscitating Xcc from dormancy, though it has been shown that Xcc, a non-sporulating bacterium, can enter the viable but non-culturable (VBNC) condition [43][44][45]. There is a study that found that Xcc could be induced into it's VBNC state in sterile soil and liquid cultures and that cupric sulfate reduced the time it took to enter this state [46]. The optimum growth temperature for Xcc has been found to be around 25-30 °C [43].

On the topic of the Abco fungus specifically, environmental factors like temperature and humidity seem to play a role in when sporulation occurs and is inhibited. One study found that naturally infected oilseed, rape, and cabbage leafs required a humidity equal to or higher than and 87% relative humidity (r.h) with

an optimal temperature of 20–30 °C [47]. Another study investigated the effects of light on Abco growth and sporulation and found that spectral variation of light did not affect the conidia (asexual spores) of Abco under in vitro conditions [48].

3

Programme of Requirements

The system has some requirements that should be met and other requirements that improve the user experience. The mandatory requirements (M.R.) are the requirements that decide whether the system operates as it should, while the trade-off requirements (T.R.) are nice to have, especially from a user standpoint. These requirements have been formulated specifically for a small-scale testing prototype that can disinfect up to approximately 15 grams of seeds at once, and would need to be revised for a larger scale model fit for commercial use.

3.1. Requirements

3.1.1. Mandatory Requirements

ID	Requirement
Performance	
M.R.1	The system must make use of UV LEDs with a peak wavelength between 250 and 270 nm.
M.R.2	The irradiation must exceed 0.5 mW/cm^2 at minimum.
M.R.3	The difference in irradiation between different locations on the plate must not exceed 25% of the average irradiation.
M.R.4	The system must be capable of disinfecting the fungus Alternaria brassicicola and the bacterium Xanthomonas campestris pv. campestris from cabbage seeds.
M.R.5	The system must support both pulsed and continuous mode operations.
M.R.6	The LED driver must be capable of delivering a peak pulse of 800 mA.
M.R.7	The LED driver in pulsed mode must be capable of outputting frequencies between 1 and 1000 Hz and duty cycles between 1 and 100% while maintaining a flat top pulse shape.
M.R.8	The system must not continue operation if the temperature exceeds safe values for the device.
Sensing	
M.R.9	The system must be capable of measuring the average temperature inside the compartment.
M.R.10	The system must be capable of measuring the average humidity inside the compartment.
M.R.11	The system must be capable of measuring the average UV irradiation reaching the seed tray.
M.R.12	The system must be capable of measuring the average ozone concentration inside the compartment.
Usability	
M.R.13	The system parameter duration must be adjustable.
M.R.14	The system parameter mode (continuous or pulsed) must be adjustable.
M.R.15	In continuous mode, the duration must be adjustable.
M.R.16	In pulsed mode, the frequency, duty cycle, and duration must be adjustable.
M.R.17	The system must display real-time sensor values and derived values, including ozone concentration, irradiance, dose, temperature, humidity, and elapsed time.

Safety and Durability	
M.R.18	The device must operate inside a closed container.
M.R.19	The maximum current through any LED must not exceed 800 mA, as specified by the datasheet, to prevent premature degradation.

Table 3.1: Mandatory Requirements

3.1.2. Trade-Off Requirements

ID	Requirement
Performance	
T.R.1	The LED driver must be capable of delivering a peak pulse of 800 mA with a rise time of no more than 5μs.
T.R.2	The system must be capable of inactivating over 99% of all existing *Alternaria brassicicola* and *Xanthomonas campestris* pv. *campestris* on cabbage seeds within 15 minutes.
T.R.3	The seed tray must be capable of spinning slowly when the device is turned on.
Sensing	
T.R.4	The system must be capable of measuring the peak LED current with an accuracy of approximately 5%.
T.R.5	The system must calculate the accumulated UV dose in real time using a UV sensor and automatically stop when the target dose is reached.
Usability	
T.R.6	The calculated expected dose should be shown as the user changes parameters in either operating mode (continuous or pulsed).
T.R.7	The current (irradiance) must be adjustable.
T.R.8	The system must display real-time sensor values and derived values, including ozone concentration, irradiance, dose, temperature, humidity, and elapsed time.
Safety and Durability	
T.R.9	The device must automatically turn off when the lid is opened.
T.R.10	The temperature of the LEDs must not exceed the maximum operating temperature of 100 °C specified in the datasheet.
T.R.11	The system shall log all run data (timestamps, mode, parameters, sensor values, and total dosage) to a microSD card.

Table 3.2: Trade-off requirements.

4

Design and Simulations

4.1. Design

The design that will be realized in this project is a high power UVC LED unit that can irradiate seeds uniformly. These will be connected to a PWM circuit that can generate any duty cycle through the microcontroller.

4.1.1. Subunits

The device is made up of four main subunits: the LED panel, the control unit driving it, the interactive unit, and the sensor panel. The LED panel is an aluminum core PCB that optimizes the layout of the LEDs for uniform irradiance and facilitates heat dissipation because the higher the current the LEDs can run on, the higher the irradiance they put out.

The control unit is responsible for driving the LEDs at the chosen duty cycle and frequency for the set amount of time. It also must connect with the sensor panel and the interactive unit to allow the user to input the desired parameters as well as show the user the current sensor measurements and the remaining time. An STM32 microcontroller is used to control all these values.

The interactive unit is made up of a small OLED display and a rotary encoder that are controlled by the microcontroller. The rotary encoder is used to input the desired parameters and the OLED display is used for user feedback in both selecting the parameters as well as observing the sensor feedback.

The sensor panel is the final subunit. It is responsible for measuring the environmental temperature, humidity, and ozone, as well as the UV irradiance that lands onto that specific location.

The subunits work together to create a prototype that shines uniform UVC light onto a seed tray at a chosen frequency, duty cycle for a chosen duration and then shows the current sensor readings and the remaining time while the device is on.

4.1.2. Chosen Materials

UVC LEDs

The most important choice in materials was the choice in LEDs. An earlier BAP project which also used LEDs for UVC disinfection concluded the maximum irradiation reached anywhere on the plate, $0.5\,\mathrm{mW/cm^2}$, showed insufficient disinfection capability. The LEDs chosen for this project where thus mainly selected based off of the flux they were capable of emitting. The final choice was the XBT-3535-UV-A130H-CH260-00 from Luminus. This is a 3.5 x 3.5 mm surface-mount device (SMD) LED that is capable of emitting up to 81 mW at a wavelength of 265 nm. It reaches this 81 mW at its rated current of 800 mA. Based on these specs and the scale of the project, 10 of these LEDs are chosen. In section 4.2.2 simulations of its irradiation will be shown.

Microcontroller

For the microcontroller an STM32G474 was chosen because of its maximum clock speed, abundance of pins and timers, a specialized high resolution timer, and its multiple ADC pins and channels. There is no doubt that regardless of what frequency range realistically could be required or what sensors might later be added that this microcontroller could handle the task. Unfortunately during one of the tests the microcontroller was fried and had to be replaced, in the meantime an STM32F411 that was on hand was used for the remaining tests. Because the frequency and duty cycles used during testing did not require extremely fast clock speeds this microcontroller performed equally well.

Boost Converter

The boost converter was chosen based off of the required voltage to drive the LEDs and the available lab power supply of 20V. As will be explained in further detail in Section 4.2.1, the LEDs were arranged in two parallel strings to reduce the required voltage from the supply but still required a boost converter that could supply at least 35V. A cheap and simple solution is provided by the XL6019 boost converter module. This DC-DC converter requires an input voltage of 3-40V and can output 5-45V with an efficiency of up to 94% and a maximum current of 5A.

Other PWM Hardware

Most of the other components that are needed for the PWM circuit responsible were chosen based on their datasheet specs and their availability in Tellegen Hall. In certain cases, some small adjustments were made to the design of the circuit to accommodate the components that were available. This will be covered in more detail in Section 4.2.1.

Sensors

The sensors were chosen based on what would be most relevant for the project. A combined temperature and humidity sensor is chosen to keep track of environmental factors that might affect the seeds or the dormancy of the pathogens. And ozone and UV sensors are included to give an indication of the effectiveness of the plasma and UV generation.

Seed Tray

To choose the tray the seeds are placed on a configuration made for either maximum reflectivity or maximum transmission. Because the total loss of transmission of UVC light due to the plasma plate between the LEDs and the seeds, wasn't certain to be low enough, the tray material was chosen for maximum transmission. In this way the option to place the UVC panel below the seed tray remained available in the case that the light being blocked would be too significant. To keep the transmission rate maximal while keeping the cost within budget, quartz glass was selected.

4.2. Simulations

To begin assessing the requirements for the hardware some calculations and simulations were done. The most important parts that needed to be assessed were the components required for the range of frequencies and duty cycles being targeted as well as the proper layout of the LEDs to maximize uniformity.

4.2.1. PWM Circuit

Calculations

The PWM circuit is responsible for generating pulses of variable frequencies and duty cycles. To achieve this the LEDs need a power supply and a transistor that can be turned on and off by a microcontroller, allowing current to flow only when the microcontroller permits it.

Because, according to the datasheet, the LEDs have a typical forward voltage (V_f) of 5.9V on top of a small dynamic resistance of approximately 1 Ω, stringing 10 of them in series would require a power supply capable of delivering over 59V. To reduce this by half, two parallel strings of 5 LEDs were used. The resulting minimum voltage required would thus become approximately 30V. Still higher than the power supplies in Tellegen can deliver, so a boost converter is added that can boost 20V up to at least 35V, and that is if the real average forward voltage of the LEDs doesn't end up being higher than the typical forward voltage specified in the datasheet.

Arranging the LEDs in parallel helps lower the voltage that must be supplied, however it does come with a downside that the current running through the parallel lines will not be exactly equal unless each line is very precisely tailored with a highly specific resistance added in series. Over time, the LEDs in the wire that has a slightly lower total forward voltage will degrade quicker due to the constantly higher current running through it. However, the greater the total amount of LEDs in each parallel line, the greater the chance that their average total forward voltage will be approximately equal. This is why the required voltage supply wasn't lowered further by splitting the LEDs into five, or even ten different parallel lines.

Moving onto the choice for transistors, a lot of different choices can be made to result in a functioning product. For example both bipolar-junction transistors (BJTs) and metal-oxide-semiconductor field-effect transistors (MOSFETs) are equally capable of taking the output from a microcontroller and switching the power supply on and off. However, because a MOSFET is more energy efficient, this was chosen. Because both N-type (NMOS) and P-type MOSFETs (PMOS) exist, each of which would result in a slighty different configuration of the PWM circuit, another choice needs to be made. Because an NMOS transistor is faster and would result in a slightly simpler circuit, this type is chosen for the design. Based on the 35V voltage supply the transistor would need to withstand as well as other specs, a choice was made between the available NMOS transistors in the lab, eventually the PSMN017-80PS was chosen. It has a voltage rating (V_{DS}) of 80V and a current rating (I_D) of 50A. Besides that, its on-resistance (R_{DS_on}), input capacitance (C_{iss}) and rise- and fall times, were minimal and well within the requirements. However, its gate threshold (V_{GS_th}) is 2-4V and requires at least 10V to fully switch. The STM32 microcontroller is only capable of delivering 3.3V, which would barely turn on the MOSFET. Thus, a gate driver component was added to the design.

The gate driver is located between the microcontroller and MOSFET, and uses its own power supply to boost the 3.3V input it receives from the microcontroller up to at least 10V. Based on the limited options available in the lab the IRS2001 gate driver was chosen. This driver can take an input of 3.3V and output 10-20V with a maximum rise and fall time of 100ns and 60ns respectively. The addition of the gate driver does result in the need for another component, a linear voltage regulator that can turn the main supply voltage down to 10V or more. In the lab there are a series of LM78XX linear voltage regulators available, one of which is an LM7812 that is capable of supplying 12V from an input up to 35V. Another linear voltage regulator is used to supply the STM32 microcontroller with its required 3.3V, the LF33AVB, which requires an input voltage up to 16V.

With the core topology established, additional circuitry is required to ensure reliable operation under dynamic load conditions. Both the laboratory power supply and the boost converter have finite transient response times and cannot instantaneously provide the current demanded by the PWM-switched LED load. To maintain a stable supply voltage during load transients, a reservoir capacitor is connected across the output.

The capacitor is sized assuming the worst-case load step, where the LED array draws 1.6 A at 35 V (800 mA per parallel string). The Tektronix PWS4205 power supply that is available in the lab specifies a voltage transient settling time of less than 400 μs following a load change, during which the capacitor must supply the difference between the instantaneous load current and the current available from the supply. Conservatively assuming that the capacitor supports the full load current throughout this 400 μs transient interval and limiting the rail voltage droop to 1%, the minimum required capacitance is calculated:

$$\Delta V = \frac{1.6A \times 400\mu s}{C}$$

The typical operating headroom for the LEDs is

$$35 - 29.5V = 5.5V$$

To maintain a margin and ensure reliable current regulation under component tolerances and operating variations, the allowable capacitor-induced voltage droop was limited to 1.5 V:

$$C = \frac{1.6A \times 400\mu s}{1.5V} \approx 427\mu F$$

A standard 470 μF capacitor was selected to provide additional design margin and to account for component tolerances, ageing effects, and variations in transient load behaviour. The resulting maximum voltage droop is approximately 1.36 V, leaving sufficient voltage headroom above the LED forward voltage.

A number of other input and output capacitors are added to some of the components discussed before according to the specifications in their datasheets. Besides this, a current limiting resistor is added in series with the LEDs to set the current that will run through them. A pull-down resistor is put between the gate and source terminals of the MOSFET to drain the MOSFET of charge when the device is off. A small resistance is also placed between the gate driver and the gate terminal of the MOSFET to prevent ringing on the gate. Calculations made for sizing these components can be found in Appendix E.1.

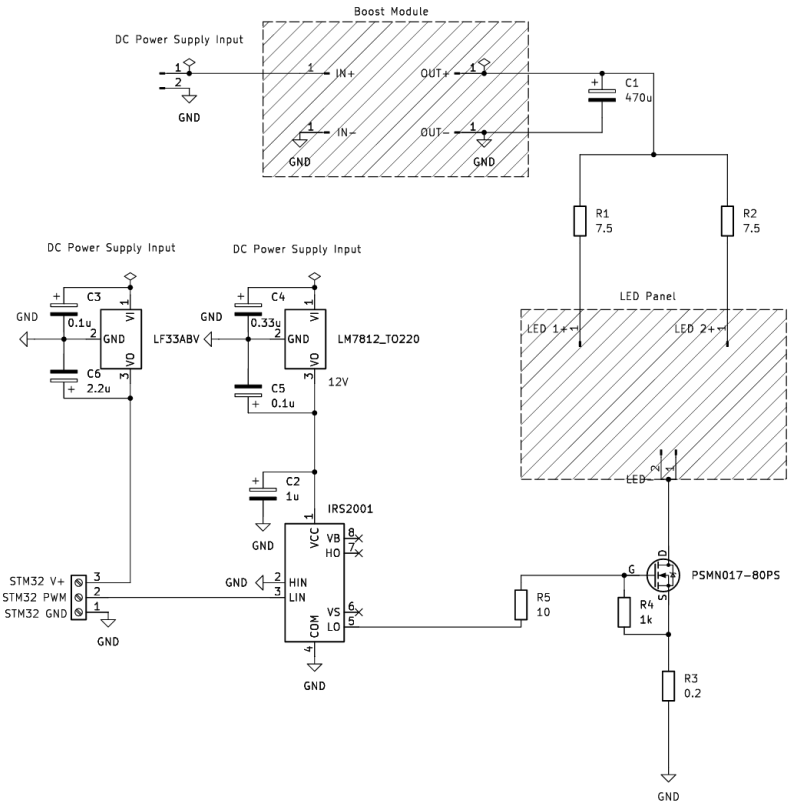


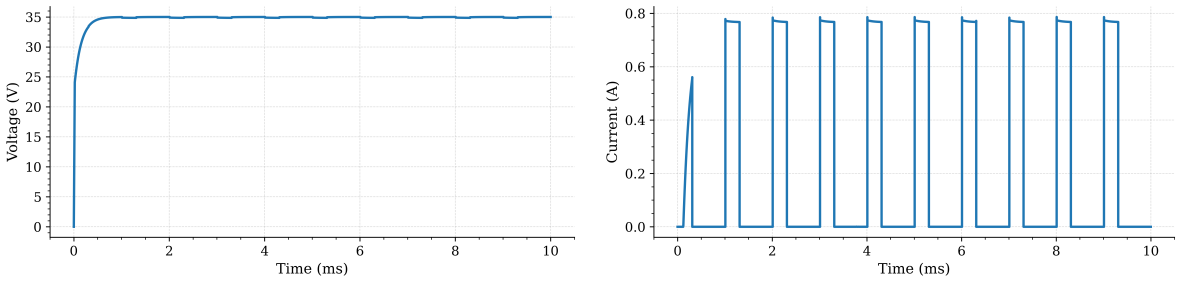
Figure 4.1: PWM circuit with specified components

The resulting PWM circuit can be seen in Figure 4.1 with the LEDs and microcontroller connections depicted through pin and screw connectors with annotations. A power analysis is done on the high-level topology in LTSpice.

Power Analysis

The high-level topography of the circuit has been drawn in LTSpice and can be seen in Figure C.1 in Appendix C. The detailed implementation has been condensed and abstracted into a roughly equivalent power model. For the power analysis an edge case will once again be used. Because the boost module was not available in time for testing, the power analysis was conducted solely keeping in mind the Tektronix PWS4205 power supply available in the lab, placed in series with a second PWS4205 to achieve the required voltage. The available components in the lab, nearest in value to the values calculated in Section 4.2.1, were measured with a multimeter so that the analysis could more accurate. On top of this an equivalent series resistance (ESR) of the reservoir capacitor is estimated to be up

to approximately 0.2 Ω. This previously unaccounted for ESR explains why the capacitor drop in the simulation is slightly higher than the one calculated manually beforehand.



(a) Simulated voltage over reservoir capacitor

(b) Simulated current through LEDs

Figure 4.2: LTSpice plots

The power analysis indicates minimal capacitor drop during it's discharge phase in Figure 4.2a, and beyond the first charging cycle the LED current remains fairly constant during it's on-phase in Figure 4.2b just below around 800 mA as is desired.

4.2.2. LED Irradiation

To ensure effective disinfection without damaging the seeds, they must be irradiated uniformly across the entire tray. This is to ensure that after testing, if pathogens are found it is not due to under-irradiated seeds. In general the seeds need enough radiation to be disinfected, but not so much they are killed themselves. While the precise upper threshold for safe exposure to UVC is not yet determined, it is known a radiation of $0.5\,\mathrm{mW/cm^2}$ is too low. To verify if the selected LED actually provide a sufficient dosage and to determine their optimal placement for maximum uniformity, optical simulations were done in Python.

First, a simulation of a singular LED was conducted to see its specific irradiation pattern. For a traditional lightbulb, all the light gets evenly or about evenly dispersed in all directions, but LEDs are directional Lambertian emitters. This means they focus most of their optical power on the center and as the viewing angle increases the irradiance drops off. To calculate the rate of this drop-off, the Lambertian order (m) is determined using Equation 4.1:

$$m = -\frac{\ln(2)}{\ln(\cos(\theta_{1/2}))} \tag{4.1}$$

The Lambertian order describes the beam profile, with an m value of 1 representing a wide spread distribution, and a very high m value indicating a highly focused beam in the center, imaging a laser. With this order the radiant intensity (I) can be found at any angle (θ) by using the peak axial intensity I_0.

$$I(\theta) = I_0 \cos^m(\theta) \tag{4.2}$$

To relate the peak intensity to the total optical power, denoted as P, the intensity is integrated over a hemisphere:

$$P = \int_0^{2\pi} \int_0^{\pi/2} I(\theta) \sin(\theta)\, d\theta\, d\phi \tag{4.3}$$

Solving this integral yields:

$$P = 2\pi I_0 \left(\frac{1}{m+1}\right) \tag{4.4}$$

Rearranging Equation 4.4 allows us to calculate the peak axial intensity directly from the LED's rated optical power:

$$I_0 = \frac{P(m+1)}{2\pi} \tag{4.5}$$

Finally, the light on the tray's surface can be calculated with the formula for flat surface irradiance (E), Equation 4.6. It accounts for the viewing angle and the inverse-square drop-off:

$$E = \frac{I_0 \cos^n(\theta)}{d^2} \tag{4.6}$$

Based on these mathematical principles, a script was written to simulate the system (the complete code can be found in Appendix B.1).

As shown in the single LED simulation graph (Figure 4.3), the maximum center irradiance reaches approximately $2.25\,\mathrm{mW/cm^2}$. This successfully satisfies the baseline project requirement of staying above $0.5\,\mathrm{mW/cm^2}$.

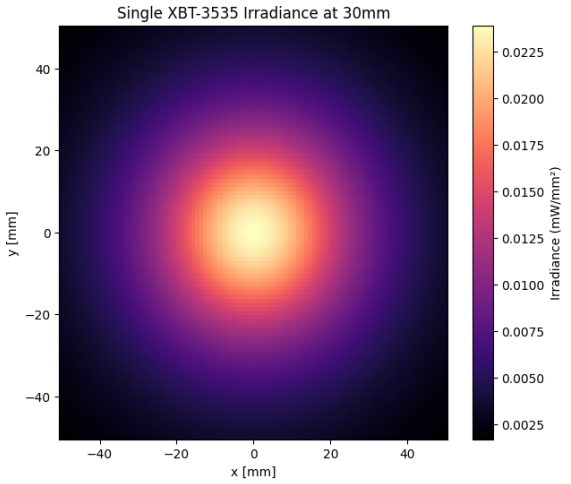
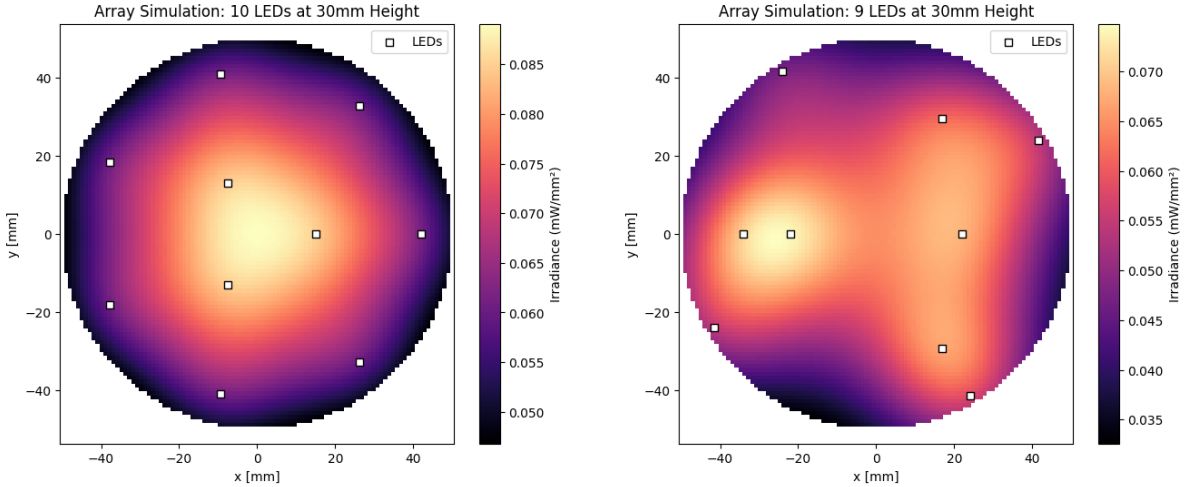


Figure 4.3: Simulation of a single LED

To find the optimal layout for all of the 10 LEDs, the simulation was expanded, with a goal to find the best pattern for a uniform irradiation across the tray. For this, several different ways were pursued to find the best approach. One way was to manually put in the coordinates and see the calculations on the maximum irradiance and the uniformity, this of course can take quite some time so an extra module was added to let the simulation determine the best position. It was done by creating two or three rings and assigning an amount of LEDs to each ring. The simulation would then find the optimal radii and where on the ring the LEDs should be placed. Since this causes some restraints in placing the LEDs another simulation was made that completely randomizes the placement. However to truly go to each possible configuration the code would have to run tens of thousands of simulations. Since it never went through all combinations, the configurations it gave were not symmetrical, which did not align with getting an actual uniform distribution. So the initial ring configuration was kept. Simulations of both can be seen in figure 4.5



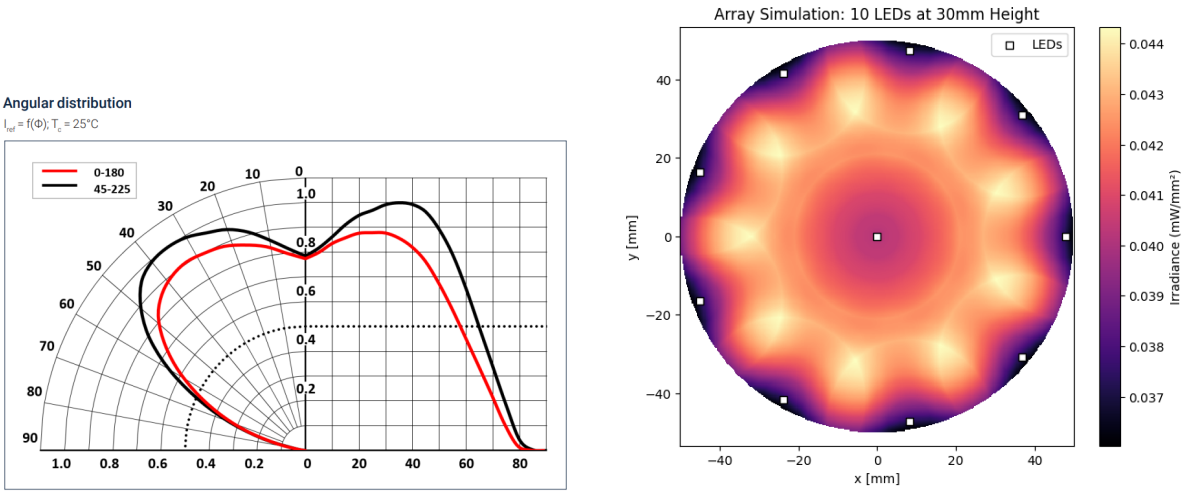
(a) First LED simulation.

(b) Second LED simulation.

Figure 4.4: Examples of LED placement simulations

An important finding was that the irradiation calculations were not completely correct. Although the previous calculations give a pretty close estimate of how the LEDs irradiate, a better way to approach is using the angular distribution graph that can be found in the data sheet of the chosen LEDs. The actual pattern is called a batwing distribution, here the highest intensity does not lie in the center but

at an angle of $30°$ to $45°$. This actually helps a lot with creating a more uniform distribution since, the light now travels more at an angle and will reach the outer edges of the tray. The precise Batwing distribution of the chosen LEDs can be seen in figure 4.5a, here it is seen that at a $0°$ the LED emits 80% of its power, while at $35°$ it radiates at its full power. Using both the radii optimization and the batwing distribution an optimal layout was found with one LED in the centre and nine of them in the outer ring, as can be seen in the figure 4.5b. This gives a uniformity error of 4.13%.



(a) Badwing **(b)** Best LED placement

Figure 4.5: Examples of LED placement simulations

The final challenge for the LED simulation is that they do not directly shine on the tray below. Instead the plasma unit from the other subgroup will be right in between, at a height of $5mm$ above the seeds. To account for this, an object was added to the simulations with the dimensions of the plasma panel. The plasma panel was made with the idea that the UVC should still get through it, by implementing glass capillaries to create a dielectric barrier. This panel however creates a lot of shadows on the tray and creates a non-uniform distribution. To get around this, the tray with seeds will be slowly rotated. With these rotations each ring of the tray needs about the same irradiation over the testing time. Running simulations to find the best layout while the tray is there lead to very high uniformity errors, still the best one found was having the same layout as suggested in figure 4.5b. Running simulations gives a 16.89% error, while preferably the error should be below 5%, the border of the plasma tray causes a ring on the tray to be significantly less irradiated than the others, as shown in figure 4.6.

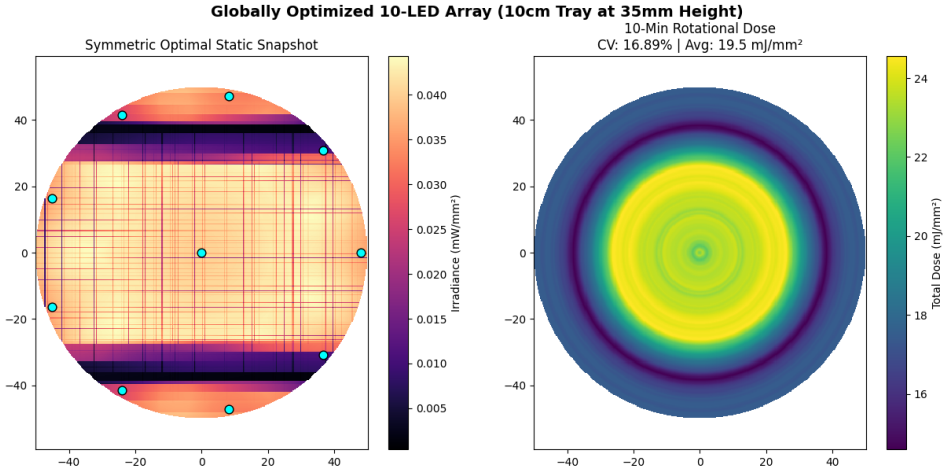


Figure 4.6: Caption

5 Hardware and Firmware

5.1. PCB Design

A custom PCB design is created for the LED panel using the software Kicad. This is necessary when pushing the LEDs to their limit of 800 mA because this amount of current requires substantial heat dissipation and the LEDs are small 3.5 x 3.5 mm SMDs that make it difficult to impossible to solder them onto a generic perfboard with wires without the joints coming loose at almost any continuous current because the pads for the three terminals are fractions of a millimeter to slightly over one millimeter wide.

5.1.1. LED Panel

The design for the LED panel is based on the calculations and simulations done in Section 4.2.2. This layout has nine LEDs placed in a circle with radius 5 cm and one LED in the center. Because the LEDs need to be placed in two parallel strings as was explained in section 4.2.1, the schematic of the LED panel is relatively straight forward and can be viewed in Figure C.3 in Appendix C.2. A single 2-pin screw connector is added to the LED configuration to allow wires to connect the anode-side node to a power supply and the cathode-side node to the NMOS transistor. The resulting PCB design of the LED panel can be seen in Figure C.4 in Appendix C.3.

3D Overview

The final design overview can be seen in Figure 5.1 with LEDs and connector already soldered on:



(a) LED panel alone

(b) LED panel with soldered components

Figure 5.1: LED panel PCB design overview

Material

When ordering the board, the option to manufacture it from aluminum instead of the standard FR-4 should be utilized as this helps increase the heat dissipation. Furthermore, the soldermask should only be applied to one side of the board so that the aluminum core can be connected to either the air or an external heatsink to further increase heat dissipation.

Actual Testing Board

Because of time constraints surrounding the project, the time it takes for biological lab results from treated seeds to return, and uncertainty in what the LED layout would definitively be until some weeks into the project, there would be no time to wait for a custom PCB to arrive. Thus an available generic SMD prototype board from the lab was used instead. This board is covered with equally spaced padded lines in between which the anodes and cathodes of the LEDs could narrowly be made to fit. However, unfortunately this didn't leave any room for the small heatsink pad on the LEDs to be connected to its own pads and would instead be connected to either the anode or cathode pad, which means the average current needs to be kept low enough to not overheat despite the lack of a heatsink. This unfortunately means the LEDs cannot be pushed to their rated current of 800 mA and would instead severely be limited to prevent overheating.

5.2. Coding

Both the main and interim STM32 microcontrollers were programmed in C in the STM32Cube ecosystem with largely the same code but some slight differences in details (e.g. pinout). The STM32CubeMX program was used to create the boilerplate code for the chosen pins and the programming was done in the STM32CubeIDE.

5.2.1. PWM

The adequate functioning of the PWM code requires accurately setting the registers of the timers. There are three main registers that need to be set as well as the clock configuration for accurate timing.

The STM32G474 and the STM32F411 are both set to their maximum clock speeds of 170 MHz and 100 MHz, respectively. This is to minimize latency from background interrupt handling.

The prescaler (PSC) register determines the clock speed of the chosen timer in such a way that:

$$Clk_{timer} = \frac{Clk_{system}}{PSC + 1}$$

With a system clock speed of 170 MHz, setting PSC to 169 results in a timer clock speed of 1 MHz. Fast enough to comfortably output a 1% duty cycle pulses of the fastest frequency within operation bounds of the device, 1 kHz. When the system clock speed is 100 MHz (for the STM32F411), the PSC register is set to 99 to achieve the same result.

The auto-reload register (ARR) can then be set to determine the frequency by determining the value the timer will count to before reloading. Based on the value of the PSC register, a frequency of e.g. 1 kHz can be configured by setting this register to 999.

Lastly the pulse register determines the length of the pulses and thus the duty cycle. This register also depends on the value of the PSC, and to achieve a pulse duration of e.g. 100 μs with the current parameters, this register can be set to 100.

Once these configurations are made in STM32CubeMX, the C code that controls the microcontroller is largely the same besides the choice in pins and timers, depending on availability.

5.2.2. Sensors

Temperature & Humidity

The temperature and humidity sensor HDC2080 can read the humidity with an accuracy of $\pm 2\%$ and the temperature with an accuracy of $\pm 0.2^{\circ}$ C. It can be powered by the 3.3 V rail from the microcontroller and works on the I^2C communication protocol, which is the same as the display that will be further elaborated in section 7.3b. It can be connected directly to the designated I^2C pins on the microcontroller, and both the humidity and the temperature data are sent over this same shared bus. Because it is a shared bus external pull-up resistors, which are typically $4.7k\Omega$, must be added to the SDA and SCL lines to keep the signals clean, as the microcontroller internal pull-up is often not strong enough. The ADDR pin is connected to GND to lock in the hardware address of 0x40. When coding for the microcontroller, the standard library needs the address to be shifted left by one bit.

UV

GUVA-S12SD is an analoge UV light sensor. Unlike the digital HDC2080 it works on a contiuous analog signal that scales linearly with the intensity of the UV light hitting it. It runs on a VCC of $3.3V$, which matches the operating voltage of the microcontroller. The output pin connects directly to one of the microcontroller's internal Analog-to-Digital Converter (ADC) pins, which reads the raw analog voltage and turns it into digital steps. The UV sensor is meant for solar radiation, meaning it will be strongest for UVA and UVB light, but will not be as sensitive to UVC. To account for this reduced sensitivity at our $275nm$ wavelength, scaling must be applied in the software. Instead of calculating a generic solar UV Index, we convert the measured voltage directly into optical power density (irradiance) using the scaling equation:

$$E_{uvc} = V_{sense} \times C_{scale}$$

$C_{scale}$the spectral scaling constant, for typical $275\,\mathrm{nm}$ UVC sources scale factor is roughly 10 to 12.5. By multiplying the voltage by this constant, it counterbalances the sensor's lower UVC sensitivity. This means an output of $0.1V$ corresponds to a real physical footprint of roughly $1.0mW/cm^2$ to $1.25mW/cm^2$ of deep UVC light on the tray surface.

Ozone

The ozone sensor is an MQ-131 mounted onto a breakout module and can be read through either its analog or digital output pin. It also needs to be powered by 5V DC which requires another linear voltage regulator. Reading it is fairly straightforward. It needs to be connected to any of the microcontroller's available ADC channel pins and its output voltage increases linearly with the ozone concentration in the environment. Because the sensor can output up to 5V, it needs to be connected with the STM32 through a simple voltage divider circuit with output resistor being two times the size of the input resistor.

5.2.3. Display

The display is integrated into the setup to provide a real-time visualization of all sensor measurements during testing. It displays the environmental and radiation values received by the microcontroller, serving as the primary user interface. In later stages, the display operates in combination with the rotary encoder, allowing the user to manually adjust the PWM duty cycle to control the UVC array.

Rotary Encoder

A rotary encoder is used for changing values and selecting choices. It is capable of turning infinitely in either direction and clicking. It has five pins: GND, VCC, $S1$, $S2$, and KEY. KEY is used to detect the button clicks. The pins $S1$ and $S2$ are used to detect both the direction of the rotation as well as the number of ticks turned in that direction. This works because the pins are connected to two channels whose terminals lie spaced slightly apart from each other on top of a conductive circular disk with equally spaced non-conductive gaps that is attached to the rotor. When the rotor spins, one of the two pins will touch the conductive pad slightly sooner than the other and also leave the conductive pad slightly sooner. The polarity of the phase difference in the pulses that are conducted by both pins gives the direction of the rotation, and the number of pulses with that phase difference gives the number of ticks in that direction. A lot of the complexity of reading and translating these signals to directions and ticks can be abstracted away by setting the microcontroller pins that connect to $S1$ and $S2$ to encoder mode.

6 Testing

With the design and simulation phases completed, the project moves to the testing and validation phase. This chapter handles all the steps taken to make a physical working UVC setup too integrating with the other subgroups and testing on the seeds. First the structured steps to get the UVC set-up working are described, then the integration with the plasma and HV-group, and all the test that where performed.

6.1. Individual Subsystem Testing

Before the UVC set-up can be integrated with the other sub-groups it was essential the full set-up was completely working on its own. Since designing circuits often can lead to a lot off debugging before it is fully functional, having to find these faults in a big testing set-up can be hard and time-consuming. Starting testing small individual parts will save time.

6.1.1. Pre-Testing

Since it is essential to know the actual physical specifications of the LEDs for proper functioning, the forward voltage V_f of the LEDs is measured. Measuring the forward voltage over a LED or other diode can be done with a multimeter by using the diode testing setting and connecting the red probe to the anode side and the black probe to cathode side of the LED. The multimeter applies a small current over the LED, which will make the LED turn on just enough to show a bit of light. On the display, it will show the forward voltage drop over the LED. However the UVC LEDs have a high turn-on threshold that the multimeter can not provide, namely around $5.3V$ to $5.9V$ as stated in the data sheet. So to find the V_f, the LED is connected to a resistor that can limit the current to ensure the LED is not destroyed. To find the correct value of the LED Ohm's law can be used:

$$R = \frac{V_s - V_f}{I_f}$$

Here V_s is the source voltage and I_f the rated current that can be found in the data sheet. This results in a resistor of 4.625Ω, placing it in series with the LED now gives the set-up to find the V_f over the LED, which is indeed the predicted $5.3V$. However due to the high current, the LEDs heat up, which causes their internal resistance to drop and, in turn makes their forward voltage lower as well. Since UVC can be harmful to the human body and is especially dangerous to look at, testing with the LEDs was done at either a safe distance or with a plastic box over the setup. Since UVC is not very strong and decreases in strength quadratically, this was sufficient.

The circuit design mentioned in section 7.3b for the PWM was soldered and tested first without the LEDs. This was done by replacing the LEDs with a resistor, since a proper input is needed at the drain pin of the MOSFET for it to turn on. Together with an oscilloscope, it was checked whether the circuit functioned correctly by checking the PWM pulse at several points.

With the LEDs being SMD, meaning they are supposed to be soldered onto a PCB, but there not being a specific PCB for the perfect layout yet, the LEDs where connected to each other by wire. These two

chains, each consisting of 5 LEDs, were then connected to the tested PWM circuit; thereby, it was concluded that the calculations done for the series resistor and the forward voltage were correct. The set-up was now ready to start testing on seeds. To do so the strings of LEDs where pinned on a small plate in the configuration seen in figure 4.5b.

The first test with this setup did cause a significant issue. Due to the LEDs becoming really hot and there being tension on the wires connecting them, the connections on the pads broke, which led to an open circuit. Since the designed PCB seen in section 7.3b would not arrive on time for testing, another solution was found, a custom physical PCB. This allowed the LEDs to be mounted securely and the wires to have little to no tension on them. An added bonus was that the heat sink pad of the LED was now also connected, so some of the heat could be dissipated. Now the first seeds could be tested on with just UVC.

6.1.2. Testing Parameters

Determining the exact bounds for effective seed disinfection involves a lot of unknown variables, as optimal irradiation dosages and exposure durations can vary significantly depending on the target seed and the specific pathogens present. Additionally, because this project evaluates the efficacy of continuous-wave radiation versus high frequency pulsed, it also adds extra possibilities for testing. Since the person who will inspect the seeds after disinfection has limited time to test, not every combination of tests can be executed. The experiment will be evaluated at duty cycles of 30%, 50%, 80%, and a 100% continuous-wave. The exposure duration across all initial tests was held constant at an interval of 5 minutes.

6.1.3. UVC disinfection

For the actual testing, the full setup was placed in a configuration as close as possible to how it should be in the final integrated setup, which is further detailed in 6.2. For each of the individual tests, 15 gram of seeds were used, following recommendation from the person who will perform the final tests on the seeds. During the 30% and 50% duty cycle tests, everything went well with testing,and the current pulled during both tests was exactly as expected. For the 80% and continuous tests, however the set-up sometimes stopped working. Through the camera that was filming the active LEDs, it could be seen that one of the parallel strings would start burning brighter than the other one, and after a while, the circuit would completely stop working. After each time this shutdown happened, an inspection revealed no shorts or open circuits. However, applying some solder on the wire connections between the LEDs did consistently solve the issue.

It was later found this can be explained by a by a phenomenon called incomplete inter-metallic bonding. Initially everything is connected properly, but due to the internal heat sink pad of the SMD LED package being connected directly to one of the active electrical pads (either the anode or the cathode), that pad now had to handle double or even more heat. Under high-power operation, the severe heat caused expansions on the LED side but not on the PCB, creating a micro-tear that is completely invisible to the naked eye. This crack however is big enough to stop the current from passing through when they system is turned on. By applying fresh solder, the micro-crack was fixed and bridged the gap, allowing the circuit to work again on the next try.

Later on, more validation tests with only the UVC setup were performed, since the full integrated setup would eventually need to run for longer durations than the 5-minute intervals that were initially tested. These expanded trials consisted of running all the previously mentioned duty cycle tests for a full 10 minutes, as well as a successful redo of the 5-minute continuous test that had failed before. With the connections structurally reinforced, the setup was now capable of running for 10 minutes at duty cycles of 80% and 100%. Additionally, the 30% duty cycle was run for extended durations of 15 and 20 minutes, as this would also be required in the fully integrated system configuration. Finally, the setup was tested with the unpowered plasma panel placed in between the light source and the seed tray to isolate and observe the exact physical shadow patterns created by its dielectric barrier structures.

6.2. Integration

6.2.1. Subunits

Finally, all subunits of the UVC device were put together. Now, the device can be controlled in real-time, and the sensor values and remaining time can be read from the display. A picture of the test set-up can be seen in 6.1, and more pictures are also shown in Appendix C.4,

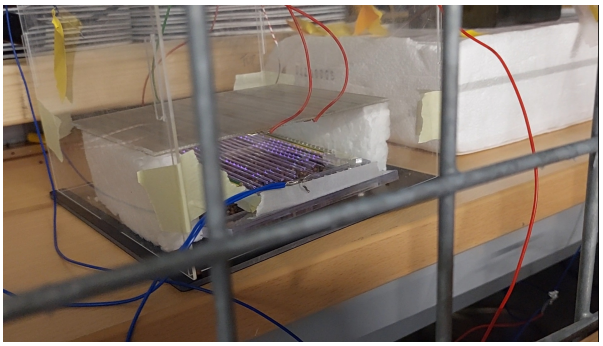


Figure 6.1: Working plasma and UVC panel disinfecting seeds

6.2.2. Plasma and HV source

To integrate our setup with the plasma and high-voltage subgroups, all further system trials had to be executed in the Else Kooi High Voltage Laboratory. The UV panel was placed at a vertical offset exactly 3 cm above the plasma panel. This clearance was safely calculated through simulations in COMSOL by the plasma group, to ensure the UV panel could operate without any risk.

Using this combined setup, full integration tests were performed at intervals of 10, 15, and 20 minutes. The integrated testing sometimes had to stop early due to localised dielectric breakdown on the plasma panel side or the LED side not working due to the as previous mentioned pad problem or the gate driver breaking. Despite these challenges, all tests were successfully executed on both dry and pre-moistened seed samples. The wet seed were specifically included in the testing to evaluate the dormancy and if it would influence the results of the set-up.

7

Results

7.1. The Banana Test

The banana test was performed as a low-cost, simple way to get an indication as to whether the quartz plate was indeed made of quartz and whether the transmission rate of UVC through it would be sufficient to justify possibly moving the UVC panel below the seeds so the plasma panel would not need to be in between the seeds and the LEDs [49].



(a) Irradiated banana

(b) Comparison: untreated banana from the same batch

Figure 7.1: The banana test results

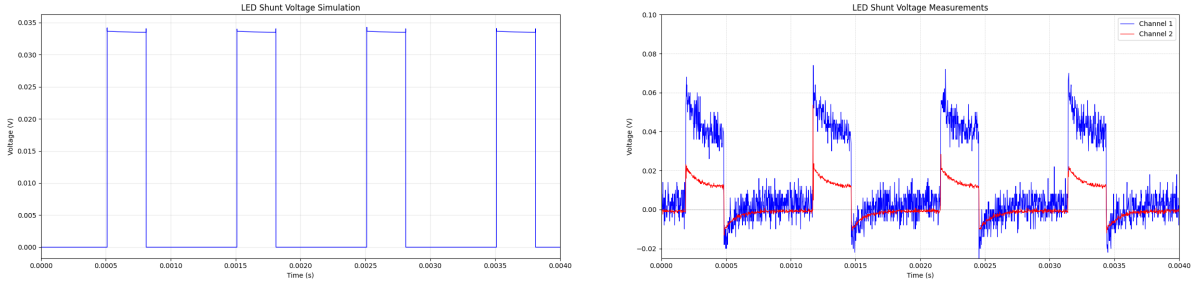
The treated banana was covered with (UVC blocking) aluminum foil everywhere except for either the bottom side of the banana during one round of irradiation, or the stem side of the banana during another round. The banana stem was irradiated without anything in between the banana and the LEDs, while the banana bottom was irradiated with the quartz glass placed in between. On the left side of Figure 7.1, the treated banana can be observed 2 days after treatment, while on the right side a banana from the same batch can be seen that was left untreated and photographed on the same day. Both show some random dark spots and browning due to aging and handling and should not be attributed to UVC irradiation either way. Besides, the randomized browning, there are also two faint borders next to more homogeneous dark patches that can be made out on the treated banana. Two of these homogeneous patches appear on the banana, one on the stem side and one on the bottom side. It appears the patch near the bottom side is slightly larger and perhaps slightly darker than the one near the stem side.

7.2. LTSpice Verification

To verify the LTSpice simulations, a 0.2 Ω shunt resistor is placed in series with each string of LEDs so the current going through each individual string could be measured. These shunts are connected to a two-channel oscilloscope to measure the difference in currents going through both strings. Besides this, the capacitor drop was also measured. For all these verification measurements the most

common testing parameters were used: 30% duty cycle, 1 kHz frequency. Because the LEDs could not be connected to a custom PCB during testing, the target peak current through the LEDs had to be decreased from 800 mA to approximately 165 mA to safely run for the full duration of all tests. Thus some other parameters like the power supply voltage also had to be altered in the LTSpice simulation. The schematic of this altered LTSpice design can be found in Figure C.2 in the Appendix.

In Figure 7.2, the left side shows the LTSpice simulation for the LED current, and the right side shows the measured LED currents. As can be seen here, the two LED strings do not have an equal current running through them. One of the two parallel lines has approximately $\frac{0.05V}{0.02\Omega}$ =250 mA peak current while the other has approximately 100 mA peak current. The simulated peak current for both lines is 175 mA which is approximately the average of the two measured peak currents. As shown in Figure 7.2b the measured LED voltages on channel 1 appear noisier than those on channel 2. This apparent increase in noise is likely influenced, at least in part, by different voltage-per-division settings for the two channels.

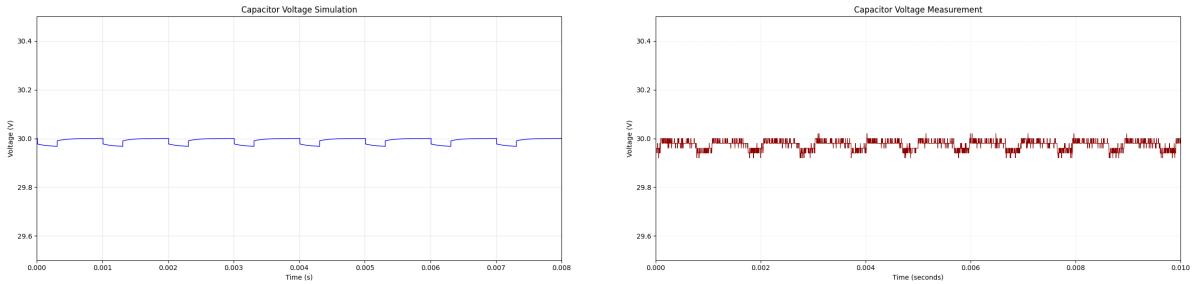


(a) Simulated LED currents

(b) Measured LED currents

Figure 7.2: LED currents: simulated vs measured

In Figure 7.3, the left side shows the LTSpice simulation for the capacitor drop with the same parameters as the eventual measurement, and the right side shows the measured capacitor drop. The measurement and the simulation seem to be fairly consistent with each other, besides the additional noise on the measurement.



(a) Simulated capacitor drop

(b) Measured capacitor drop

Figure 7.3: Capacitor drop: simulated vs measured

7.3. PCR and agar tests

Two biological assays are used to assess disinfection performance agar plate testing and quantitative polymerase chain reaction (qPCR) testing [50] [51]. The agar plate tests are evaluated after nine days because the seeds are placed onto a nutrient-rich agar plate and maintained under controlled temperature conditions to create an optimal environment for microbial growth. The microbes then need time to proliferate into colonies that can be observed and quantified with the naked eye.

The PCR tests however yield results within hours. This test artificially increases the DNA of the pathogens by separating the two strands of the double-helix with heat, opening them up to DNA replication by adding primers, small molecules that will bind to specific parts of the DNA. Polymerase

molecules are added that can bind to those primers and start duplicating parts of the DNA into more copies [52]. The quantity of DNA can then be measured by the PCR machine by using techniques such as making the DNA fluoresce and measuring the light emitted. A substance called propidium-monoazide (PMA) can be added to exclusively bind with cells with damaged membranes so these dead cells cannot bind to the polymerase molecules and wont be amplified [50] [53] [54]. This leaves only living cells to be copied and quantified by the machine, which gives a better indication of how effective the disinfection treatment is, although the slower agar tests remains the gold standard for evaluating the true viability of the pathogen [55].

Although the true formula to calculate the disinfection rate of the various tests is more complicated, for these tests we were instructed to use a minimum Cq of 36 as a measure of the eradication of the Xcc bacterium and a minimum Cq of 32 for the Abco fungus.

7.3.1. UVC solo

The first tests were done looking at the effects of UVC alone. The seeds were irradiated with a frequency of 1 kHz for 5 minutes at three different duty cycles: 30%, 50%, and 80% with an average peak current of ≈165 mA.

PCR

The results of the PCR tests can be seen in Figures 7.4 and 7.5. Each sample was split in half before being tested so that the test could be run twice on different seed batches from the same sample to increase the precision of the results. As can be observed from these results, even the 5 minute UVC treatment at the lowest tested duty cycle and throttled current offers an improvement over the control group. The 50% duty cycle sample performed approximately equally well, while the 80 % duty cycle sample seemed to have performed slightly worse. When it comes to the Abco fungus, all tested duty cycles seem approximately equally effective and the difference between the two measurement repetitions of each treatment has also decreased. However, the samples are all further removed from eliminating the pathogen. The tables with the full numeric results can be found in Appendix D.

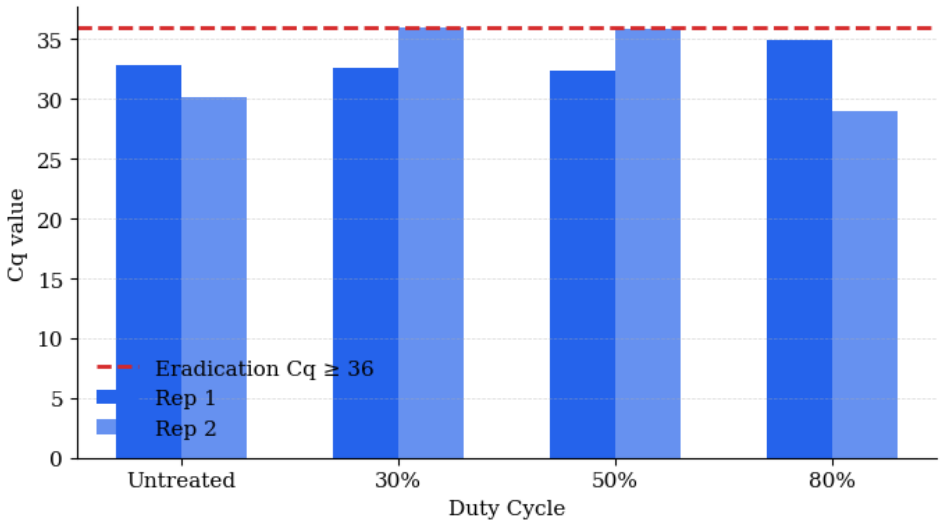


Figure 7.4: Cq values for the qPCR test with PMA for the Xcc bacterium after 5 minute UVC radiation with different duty cyles at frequency of 1 kHz and an average LED peak current of ≈ 165 mA

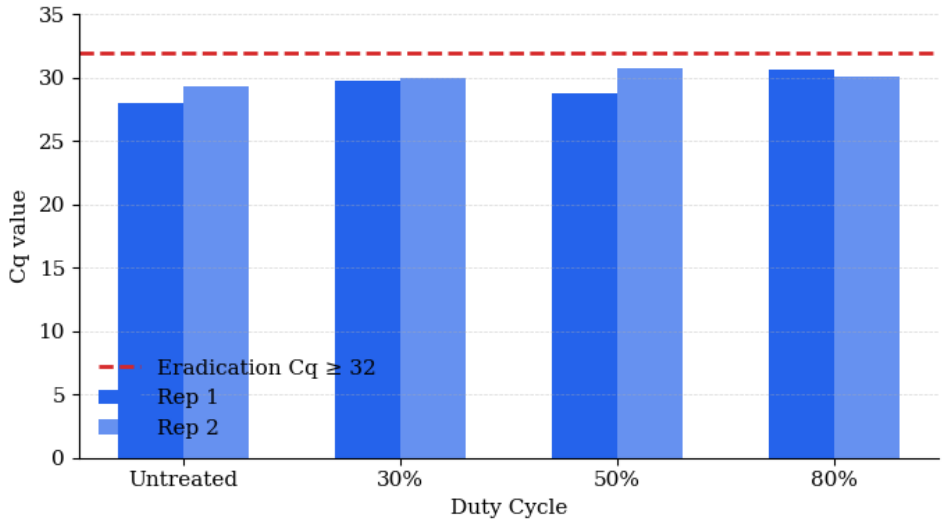


Figure 7.5: Cq values for the qPCR test with PMA for the Abco fungus after 5 minute UVC radiation with different duty cyles at frequency of 1 kHz and an average LED peak current of ≈ 165 mA

Agar

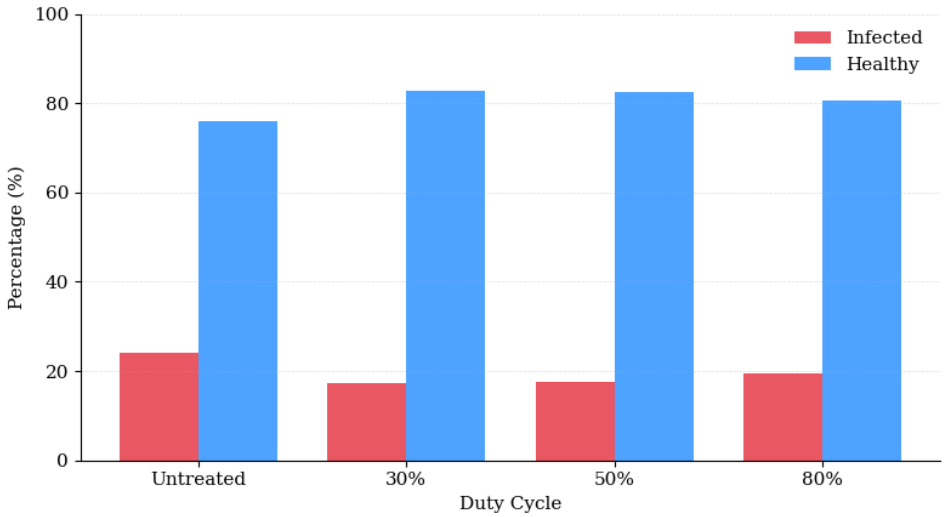


Figure 7.6: Agar plate results for the bacterium Xanthomonas campestris pv. campestris after 5 minute UVC radiation with different duty cyles at frequency of 1 kHz and an average LED peak current of ≈ 165 mA

The results of the agar test can be seen in Figure 7.6. This test only assesses the effect of the treatment on Alternaria Brassicicola. In this figure, it can be seen that all duty cycles offer a benefit when compared to the untreated sample. However none of the duty cycles have managed to fully eliminate the pathogen within 5 minutes with the frequency and current that was used during testing. With all duty cycles between 15 and 20% have not been disinfected. Lower duty cycles seem to slightly outperform higher duty cycles.

7.3.2. UVC and Plasma

The results for the combined UVC and plasma treatments will be discussed next. For all of these treatments, the UVC parameters were set to be pulsing at a frequency of 1 kHz with a 30% duty cycle and an average LED peak current of ≈ 165 mA.

PCR

The results of the PCR test for the combined UVC and plasma treatment on the Xcc bacterium can be seen in Figure 7.7. Some of the repetitions failed and were therefore excluded from the analysis. The 5 minute treatment appears to perform better than the untreated control, whereas the 10 minute treatment appears to perform worse. For the 20 minute treatment, only one PCR test was successfully completed. This test performed slightly worse than the best performing repetition of the 10 minute treatment but outperformed all other treatments and repetitions. Both repetitions of the untreated seeds also exceed the minimum Cq that indicates eradication of the pathogen.

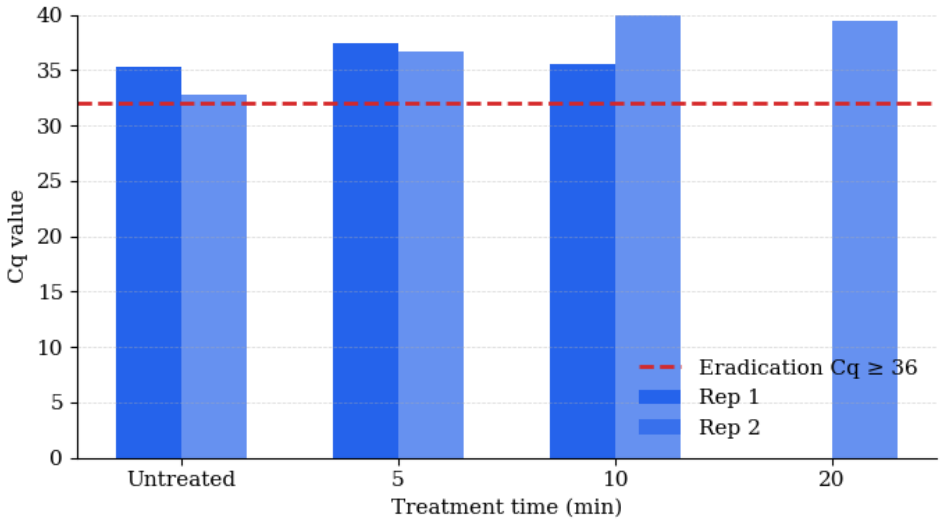


Figure 7.7: Cq values for the qPCR test with PMA for the Xcc bacterium after combined UVC and plasma treatment for different time durations. UVC frequency of 1 kHz, duty cycle of 30%, average LED peak current of ≈ 165 mA.

The results of the PCR test for the combined UVC and plasma treatment on the Abco fungus can be seen in Figure 7.8. A 5 minute treatment seems to perform slightly better than no treatment at all while the 10 minute treatment seemingly performs worse. The 20 minute treatment performs best, with both test repetitions nearing or exceeding the minimum Cq value that indicates eradication of the pathogen.

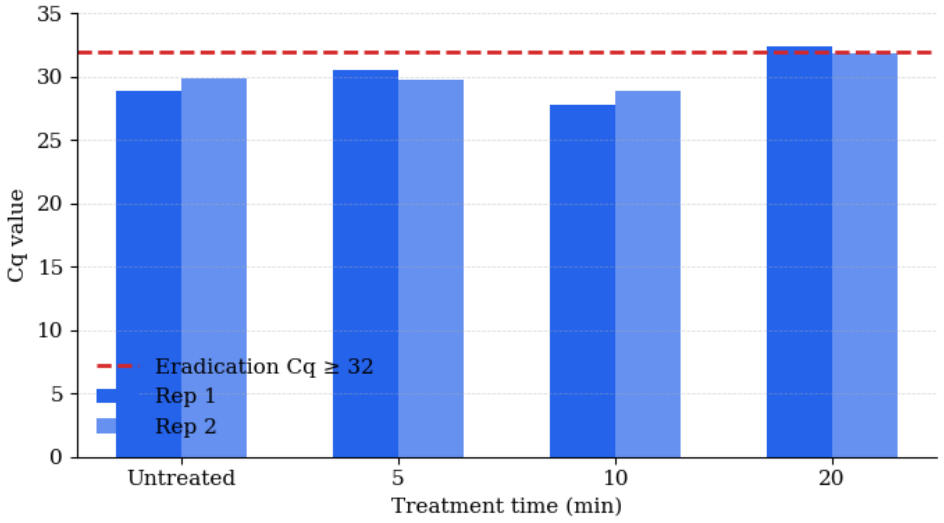


Figure 7.8: Cq values for the qPCR test with PMA for the Abco fungus after combined UVC and plasma treatment for different time durations. UVC frequency of 1 kHz, duty cycle of 30%, average LED peak current of ≈ 165 mA.

Dormancy

To attempt to break pathogens out of the dormant state they may be in before the treatment hits, an additional test was done where the seeds were sprayed with cold water minutes prior to the same treatments that were performed in 7.3.2.

The results of the PCR test for the Xcc bacterium, following combined UVC and plasma treatment on seeds sprayed with cold water, are shown in Figure 7.9. While the 10 minute treatment seems to perform better than the one that was done without spraying the seeds (see Figure 7.8), the 20 minute treatment seems to have done worse. Notably none of the results for any treatments exceed the minimum Cq to indicate the pathogen has been eradicated.

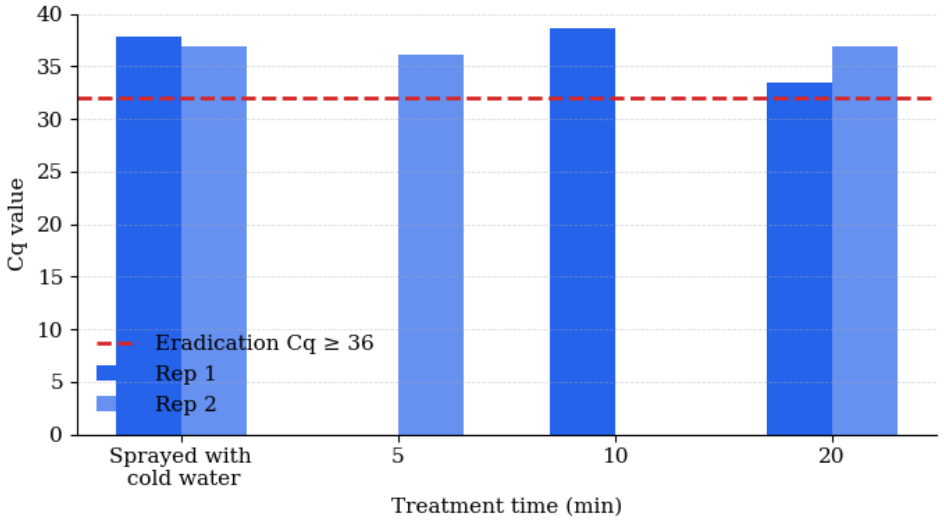


Figure 7.9: Cq values for the qPCR test with PMA for the Xcc bacterium after seeds were sprayed with cold water (to attempt to break pathogen dormancy) preceding a combined UVC and plasma treatment for different time durations. UVC frequency of 1 kHz, duty cycle of 30%, average LED peak current of ≈ 165 mA.

The results of the PCR test for the Abco fungus, following combined UVC and plasma treatment on seeds sprayed with cold water, are shown in Figure 7.10. While the 10 minute treatment seems to perform better than the one that was done without spraying the seeds (see Figure 7.8), the 20 minute treatment seems to have done worse. Notably none of the results for any treatments exceed the minimum Cq to indicate the pathogen has been eradicated.

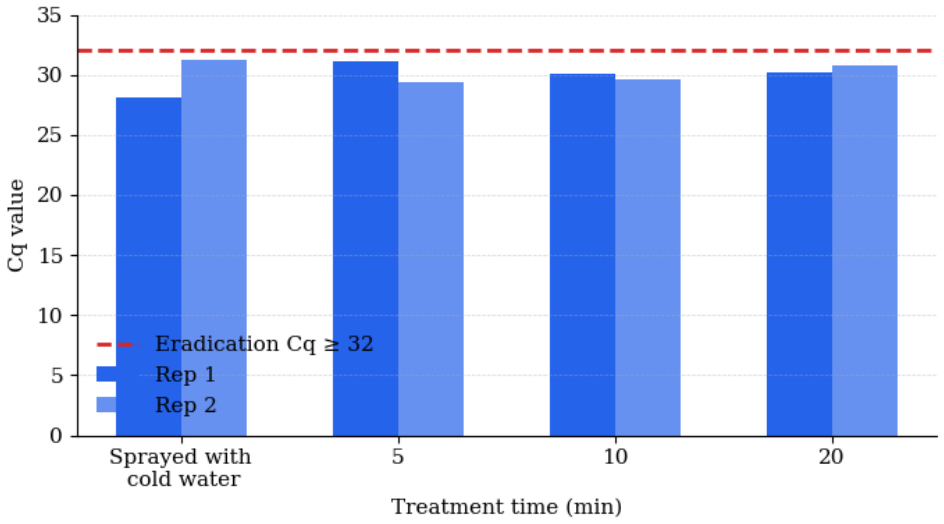


Figure 7.10: Cq values for the qPCR test with PMA for the Abco fungus after seeds were sprayed with cold water (to attempt to break pathogen dormancy) preceding a combined UVC and plasma treatment for different time durations. UVC frequency of 1 kHz, duty cycle of 30%, average LED peak current of ≈ 165 mA.

7.4. Seed Germination Test

To evaluate whether the disinfection of the pathogens didn't also have negative effects on the seed itself a short-term seed growth experiment was done where the seeds were kept in a humid environment in a window sill for five days to see how many seeds out of the total planted sprouted by then. An image of the seeds after five days is shown in Figure C.14 (Appendix). The corresponding treatment labels for each pot are indicated in the figure, and the full treatment descriptions and germination rates are provided in Table 7.1.

Treatment	Total (n)	Germinated (n)	Germination (%)	Not Germinated (n)	Not Germinated (%)
A)UVC 30%	9	9	100	0	0
B)UVC 100%[1]	9	9	100	0	0
C)UVC+Plasma[2]	8	7	87.5	1	12.5
D)UVC+Plasma	7	7	100	0	0

Table 7.1: Germination test results. All treatments lasted 10 minutes. UVC parameters unless stated otherwise: 1 kHz frequency, 30% duty cyle, 165 mA average peak current. [1] : UVC was continuous, average peak current 100 mA. [2] : Seeds were sprayed with cold water before treatment to find effects on pathogen dormancy.

8

Discussion

8.1. General

The banana test showed an unexpected result. It looks like the side (bottom part of banana) that was irradiated with quartz glass in between the LEDs and the banana might have a larger and slightly darker mark. It is possible this was due to the small difference in placement of the banana during the two different treatment cycles. Perhaps the side that got irradiated with quartz glass in front of the LEDs was angled up slightly closer to the LEDs or perhaps even closer towards the parallel string of LEDs that conducted more current. It is not unlikely these slight differences could have made an impact considering the part closest to the bottom of the banana seems more yellow even though it was also uncovered and placed below the LEDs during the irradiation. A more conclusive result could be found if the banana was irradiated for longer or by simply using tools like UVC testing cards.

8.2. UVC alone

Overall, the results seem promising. While none of the solo UVC qPCR samples showed both reps easily crossing the Cq threshold, it is a positive sign that they consistently came close even though the current and thus the irradiation the LEDs were putting out were severely throttled due to the lack of a heatsink. Although this wasn't considered a potential benefit to pulsed UVC beforehand, it did help allow the LEDs to run at a much higher peak current (slightly over 150 mA on average, although not evenly distributed between the two parallel LED lines) for longer durations than they could handle if they weren't pulsed. That showed during testing when the 30% and 50% duty cycle samples managed to complete the 5 minute duration without fuss while the 80% sample had to be restarted multiple times to complete the full 5 minute run in between cooling breaks. The continuous sample had to be stopped 3 minutes in because it took too long to finish. Better results can potentially be achieved in less time if the current and thus irradiance can be increased by using a custom PCB that allows for rapid heat dissipation.

One unexpected result showed that the 80% duty cycle sample seemed to perform worse overall despite delivering the highest dosage. It is possible that something went wrong during testing due to rushing, but it's not immediately clear what that might be. The total time durations were all timed and the duty cycles were all tested directly before the tests as well. It is possible that the repeated cooling breaks had a negative effect. What also stood out was that 30 and 50% seemed to be approximately equally effective, even though there is a difference in the total dosage delivered. Repeating the experiment could give indications as to whether this was due to experimental error or reflects a genuine effect.

The effects of varying the frequency or time duration were not explicitly researched due to time constraints and the amount of biological tests that could be done within those constraints, but could also yield interesting results.

Assuming the results show a genuine effect, there is a real potential for saving energy by using an

equally or more effective lower duty cycle to disinfect seeds with UVC.

8.3. Integrated Plasma and UVC

UVC and Plasma Frame

Tests were done that irradiated seeds with UVC and an (inactive) plasma frame in between at three different duty cycles (30%, 50%, 80%), but due to limited time those couldn't be assessed with biological assays, so the exact measured effect of the plasma frame on the UVC irradiation that reaches the seed tray remains uncertain. Besides the unknown effects of the plasma frame blocking UV emission from the LEDs, there is also a chance that having a high voltage electric field nearby could have a positive synergistic effect on UVC disinfection [56].

Another issue that was later discovered was that the wrong MQ-131 ozone sensor had apparently been ordered. Apparently this sensor comes in both a parts per billion (ppb) version and a parts per million (ppm) version, and the ppb version only measures up to 1 ppm, which would be insufficient to measure the large quantities of ozone the plasma subunit would create. The plasma team had a more accurate separate ozone meter on hand during the entire duration of testing though, so this didn't negatively impact any recorded results.

8.4. Future Work

Primarily, a future iteration of this project should make sure to keep the heat dissipation of the LEDs in mind and ensure that a custom PCB that helps facilitate that is ordered in time so the LEDs can be pushed to their maximum current. Besides this the effects of frequency and time duration should also be tested. There could also be benefit to researching even lower wavelength UVC for disinfection purposes (222 nm) [57]. This would especially be beneficial when it comes to safety precautions as at these very low wavelengths the UV light is less capable of harming people due to its high absorption in air [57]. Next to the safety aspect, it is known that protein damage is predominant at a wavelength of around 220 nm [27].

For a commercial large scale product disinfecting seeds, different configurations of a system that utilizes UVC and/or plasma for disinfection should be considered. For example, a tunnel system could be considered [58]. This would allow for large batches of seeds to automatically be processed without opening and closing a compartment to manually move batches of seeds.

9

Conclusion

This project successfully developed a software-controlled UVC seed disinfection system prototype. The finalized design features an STM32 microcontroller generating a 1 kHz PWM signal to drive high-power UVC LEDs. It features real-time environment monitoring, including temperature and UV irradiance, tracked via an OLED display.

The system successfully met key requirements from the Programme of Requirements, notably implementing physical shielding to safely contain UVC exposure and a startup time-delay feature to protect operators. However, because the custom aluminum-core PCB was delayed, a standard alternative PCB was used. Due to the lack of an optimized thermal footprint, the peak LED current had to be safely restricted to an average of $165\ mA$ instead of the intended $800\ mA$ to avoid thermal damage and ensure a reliable working setup.

Biological testing on agar plates confirmed that a 5-minute standalone UVC treatment significantly reduced pathogen growth compared to untreated samples, with lower duty cycles (30% and 50%) proving just as effective as higher ones while drastically reducing component heat stress. Ultimately, because standalone UVC does not achieve 100% eradication, the integrated approach combining UVC and cold atmospheric plasma is necessary to bridge the performance gap and offer a viable, chemical-free alternative to traditional pesticides.

References

[1] Jean B Ristaino et al. “The persistent threat of emerging plant disease pandemics to global food security”. In: *Proceedings of the National Academy of Sciences* 118.23 (2021).

[2] Food and Agriculture Organization of the United Nations. *World Agriculture Towards 2030/2050: The 2012 Revision*. Tech. rep. ESA/2012. Rome, Italy: FAO Global Perspective Studies Unit, 2012.

[3] Tim Searchinger et al. *Creating a Sustainable Food Future: A Menu of Solutions to Feed Nearly 10 Billion People by 2050*. Synthesis Report. Washington, DC, USA: World Resources Institute, 2019.

[4] Margaret Mullaney, Shilo Takeguchi$_{M}ullaney$, and John F. Tooker. “Sowing Uncertainty: What We Do and Don’t Know about the Planting of Pesticide-Treated Seed”. In: *BioScience* 70.5 (2020), pp. 390–395. DOI: 10.1093/biosci/biaa019.

[5] Fortune Business Insights. *Global Seed Treatment Market Size, Share & Industry Analysis, By Type (Synthetic Chemicals, Biologicals)*. Tech. rep. Pune, India: Fortune Business Insights, 2026. URL: https://www.fortunebusinessinsights.com/industry-reports/seed-treatment-market-100156.

[6] Food and Agriculture Organization of the United Nations. *Pesticides use, pesticides trade and pesticides indicators: Global, regional and country trends, 1990–2020*. Tech. rep. 46. Rome, Italy: FAO, 2022. URL: https://openknowledge.fao.org/handle/20.500.14283/cc0901en.

[7] Straits Research. *Cold Plasma Market Size, Share, Growth, Analysis, Report, 2026-2034*. Tech. rep. New York, NY, USA: Straits Research, 2026. URL: https://straitsresearch.com/report/cold-plasma-market.

[8] Grand View Research. *Ultraviolet Disinfection Equipment Market Report, 2026–2033*. Tech. rep. San Francisco, CA, USA: Grand View Research, 2026. URL: https://www.grandviewresearch.com/industry-analysis/ultraviolet-uv-disinfection-equipment-market.

[9] Inductiveload. *EM Spectrum Properties edit.svg*. Wikimedia Commons. 2007. URL: https://commons.wikimedia.org/wiki/File:EM_Spectrum_Properties_edit.svg.

[10] *Ultraviolet (UV) Radiation*. 2017. URL: https://scied.ucar.edu/learning-zone/atmosphere/ultraviolet-uv-radiation.

[11] *Visible Light*. Aug. 2023. URL: https://science.nasa.gov/ems/09_visiblelight/.

[12] 2025. URL: https://www.bfs.de/EN/topics/opt/visible-light/introduction/introduction.html.

[13] Apr. 2013. URL: https://www.noaa.gov/jetstream/satellites/electromagnetic-waves.

[14] URL: https://iuva.org/What-is-UV.

[15] R. Paschotta. *Band Gap*. RP Photonics Encyclopedia. Available online at https://www.rp-photonics.com/band_gap.html. 2019. DOI: 10.61835/9f5. URL: https://www.rp-photonics.com/band_gap.html (visited on 06/15/2026).

[16] Juan I. Larruquert. “Chapter 7 - Optical properties of thin film materials at short wavelengths”. In: *Optical Thin Films and Coatings (Second Edition)*. Ed. by Angela Piegari and François Flory. Second Edition. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2018, pp. 291–356. ISBN: 978-0-08-102073-9. DOI: https://doi.org/10.1016/B978-0-08-102073-9.00007-2. URL: https://www.sciencedirect.com/science/article/pii/B9780081020739000072.

[17] IARC Working Group on the Evaluation of Carcinogenic Risks to Humans. “Solar and Ultraviolet Radiation”. In: *IARC Monographs on the Evaluation of Carcinogenic Risks to Humans, Volume 100D: Radiation*. IARC Monographs No. 100D. Lyon, France: International Agency for Research on Cancer, 2012. URL: https://www.ncbi.nlm.nih.gov/books/NBK304366/.

[18] Tianhong Dai et al. “Ultraviolet C irradiation: An alternative antimicrobial approach to localized infections?” In: *Expert Review of Anti-infective Therapy* 10.2 (Feb. 2012), pp. 185–195. DOI: 10.1586/eri.11.166.

[19] Timothy D. Cutler and Jeffrey J. Zimmerman. “Ultraviolet irradiation and the mechanisms underlying its inactivation of infectious agents”. In: *Animal Health Research Reviews* 12.1 (June 2011), pp. 15–23. DOI: 10.1017/s1466252311000016.

[20] URL: https://www.genome.gov/about-genomics/educational-resources/fact-sheets/ribonucleic-acid-fact-sheet.

[21] Sarah Bates. *Deoxyribonucleic acid (DNA)*. URL: https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid-DNA.

[22] Navdeep S. Chandel. “Nucleotide metabolism”. In: *Cold Spring Harbor Perspectives in Biology* 13.7 (July 2021). DOI: 10.1101/cshperspect.a040592.

[23] June 2019. URL: https://humanoriginproject.com/dna-structure-function/.

[24] David S. Goodsell. “The molecular perspective: Ultraviolet light and pyrimidine dimers”. In: *STEM CELLS* 19.4 (July 2001), pp. 348–349. DOI: 10.1634/stemcells.19-4-348.

[25] Sanjeev K. Bhardwaj et al. “UVC-based photoinactivation as an efficient tool to control the transmission of coronaviruses”. In: *Science of The Total Environment* 792 (Oct. 2021), p. 148548. DOI: 10.1016/j.scitotenv.2021.148548.

[26] Kim Lundgreen. *Methods to verify organism viability in treated ballast water*. 2019. URL: https://www.researchgate.net/publication/335836178_Kim_Lundgreen_2019_-_PhD_Thesis_Methods_to_verify_organism_viability_in_treated_ballast_water.

[27] Yoontaek Oh et al. “Inactivation efficacy and mechanisms of wavelength-specific UV sources for various strains of Legionella pneumophila serogroup 1”. In: *Science of The Total Environment* 907 (Jan. 2024), p. 167781. DOI: 10.1016/j.scitotenv.2023.167781.

[28] *UV-Vis Spectrophotometry – Easy and Quick Quantification of Nucleic Acids*. URL: https://www.eppendorf.com/nl-en/lab-academy/life-science/cell-biology/uv-vis-spectrophotometry-easy-and-quick-quantification-of-nucleic-acids/.

[29] P. Soengas. *Xanthomonas campestris*. https://encyclopedia.pub/entry/13686. Encyclopedia.pub entry; accessed 2026-04-27. Aug. 2021.

[30] Joint Genome Institute (JGI), U.S. Department of Energy. *Alternaria brassicicola Genome Portal (Altbr1)*. https://mycocosm.jgi.doe.gov/Altbr1/Altbr1.home.html. MycoCosm fungal genome portal; accessed 2026-06-16. n.d.

[31] *UV Dosage Required to Kill Microorganisms*. URL: https://uv-light.co.uk/uv-dosage-required-to-kill-microorganisms/.

[32] Feb. 2026. URL: https://ultraviolet.com/microorganisms-inactivated/.

[33] Muhammad Salman Mohsin and Mariana Lanzarini-Lopes. “Why dose alone fails: Irradiance-dependent bacterial survival mechanisms challenge UV system validation”. In: *Water Research X* 32 (Sept. 2026), p. 100564. DOI: 10.1016/j.wroa.2026.100564.

[34] X.-Y. Zou et al. “Enhanced inactivation of E. coli by pulsed UV-LED irradiation during water disinfection”. In: *Science of The Total Environment* 650 (Feb. 2019), pp. 210–215. DOI: 10.1016/j.scitotenv.2018.08.367. URL: https://doi.org/10.1016/j.scitotenv.2018.08.367.

[35] J. Li et al. “Enhanced germicidal effects of pulsed UV-LED irradiation on biofilms”. In: *Journal of Applied Microbiology* 109.6 (Sept. 2010), pp. 2183–2190. DOI: 10.1111/j.1365-2672.2010.04850.x. URL: https://doi.org/10.1111/j.1365-2672.2010.04850.x.

[36] E. Flores Gallegos et al. “Continuous and Pulsed Ultraviolet-C LED on Germicidal Effect and Energy Consumption in Fresh Products: Applications in Tomatoes as a Model”. In: *Foods* 11.22 (Nov. 2022), p. 3636. DOI: 10.3390/foods11223636. URL: https://doi.org/10.3390/foods11223636.

[37] S. Wengraitis et al. “Pulsed UV-C Disinfection of Escherichia coli With Light-Emitting Diodes, Emitted at Various Repetition Rates and Duty Cycles”. In: *Photochemistry and Photobiology* 89.1 (2013), pp. 127–131. DOI: 10.1111/j.1751-1097.2012.01203.x. URL: https://doi.org/10.1111/j.1751-1097.2012.01203.x.

[38] Mark D. McDonald et al. “What is Microbial Dormancy?” In: *Trends in Microbiology* 32.2 (Feb. 2024), pp. 142–150. DOI: 10.1016/j.tim.2023.08.006.

[39] Mingwei Huang and Christina M. Hull. “Sporulation: How to survive on planet earth (and beyond)”. In: *Current Genetics* 63.5 (Apr. 2017), pp. 831–838. DOI: 10.1007/s00294-017-0694-7.

[40] Junyan Liu et al. “Viable but nonculturable (VBNC) state, an underestimated and controversial microbial survival strategy”. In: *Trends in Microbiology* 31.10 (Oct. 2023), pp. 1013–1023. DOI: 10.1016/j.tim.2023.04.009.

[41] Jun□Seob Kim et al. “Viable but non□culturable and persistence describe the same bacterial stress state”. In: *Environmental Microbiology* 20.6 (Apr. 2018), pp. 2038–2048. DOI: 10.1111/1462-2920.14075.

[42] Yongqiang Gao et al. “Bacterial spore germination receptors are nutrient-gated ion channels”. In: *Science* 380.6643 (Apr 2023), pp. 387–391. DOI: https://doi.org/10.1126/science.adg9829.

[43] H. et al. Mácha. “Killing Effect of Bacillus velezensis FZB42 on a Xanthomonas campestris pv. campestris (Xcc) Strain Newly Isolated from Cabbage (Brassica oleracea convar. capitata (L.)): A Metabolomic Study”. In: *Microorganisms* 9.7 (2021), p. 1410. DOI: 10.3390/microorganisms9071410.

[44] P. Tchato et al. “From Dormancy to Viability: The Resuscitation Processes of Viable but Non-Culturable Bacteria—A Systematic Review”. In: *Microorganisms* 14.1 (2026), p. 136. DOI: 10.3390/microorganisms14010136.

[45] P. et al. Alexandre. “Induction of the viable but non-culturable condition in Xanthomonas campestris pv. campestris in liquid microcosms and sterile soil”. In: *FEMS Microbiology Ecology* 30.3 (1999), pp. 203–208. DOI: 10.1111/j.1574-6941.1999.tb00648.x.

[46] J. I. Ghezzi and T. R. Steck. “Induction of the viable but non-culturable condition in Xanthomonas campestris pv. campestris in liquid microcosms and sterile soil”. In: *FEMS Microbiology Ecology* 30.3 (Nov. 1999), pp. 203–208. DOI: 10.1111/j.1574-6941.1999.tb00648.x.

[47] F.M. Humpherson-Jones and Kathleen Phelps. “Climatic factors influencing spore production in alternaria brassicae and Alternaria brassicicola”. In: *Annals of Applied Biology* 114.3 (Jun 1989), pp. 449–458. DOI: 10.1111/j.1744-7348.1989.tb03360.x.

[48] S. Kundu, M. Patra, and K. R. Samaddar. “Factors affecting growth and sporulation of Alternaria brassicicola”. In: *Journal of Mycopathological Research* 29.1 (1991), pp. 17–22.

[49] N.L. Wade et al. “Sunscald and ultraviolet light injury of banana fruits”. In: *Journal of Horticultural Science* 68.3 (1993), p. 409–419. ISSN: 0022-1589.

[50] Stephen A Bustin et al. “The MIQE guidelines: Minimum information for publication of quantitative real-time PCR experiments”. In: *Clinical Chemistry* 55.4 (Apr. 2009), pp. 611–622. DOI: 10.1373/clinchem.2008.112797.

[51] Yihan Zhang et al. “Enhancing escherichia coli inactivation: Synergistic mechanism of ultraviolet light and high-voltage electric field”. In: *Foods* 13.9 (Apr. 2024), p. 1343. DOI: 10.3390/foods13091343.

[52] Kary B. Mullis and Fred A. Faloona. “Specific synthesis of DNA in vitro via a polymerase-catalyzed chain reaction”. In: *Methods in Enzymology* (1987), pp. 335–350. DOI: 10.1016/0076-6879(87)55023-6.

[53] Andreas Nocker et al. “Use of propidium monoazide for live/dead distinction in microbial ecology”. In: *Applied and Environmental Microbiology* 73.16 (Aug. 2007), pp. 5111–5117. DOI: 10.1128/aem.02987-06.

[54] URL: https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/propidium-monoazide.

[55] Christian T. Meyer et al. “A high-throughput and low-waste viability assay for microbes”. In: *Nature Microbiology* 8.12 (Nov. 2023), pp. 2304–2314. DOI: 10.1038/s41564-023-01513-9.

[56] Yihan Zhang et al. “Enhancing escherichia coli inactivation: Synergistic mechanism of ultraviolet light and high-voltage electric field”. In: *Foods* 13.9 (Apr. 2024), p. 1343. DOI: 10.3390/foods13091343.

[57] Ewan Eadie et al. “Far-UVC (222 nm) efficiently inactivates an airborne pathogen in a room-sized chamber”. In: *Scientific Reports* 12.1 (Mar. 2022). DOI: 10.1038/s41598-022-08462-z.

[58] *UVC□LED tunnel: Cheaper, more consistent and more sustainable disinfection for hospital pharmacy*. Apr. 2026. URL: https://natdis.eu/advice-and-meeting-your-needs-with-respect-to-uvc-solutions/uvc-led-tunnel-cheaper-more-consistent-and-more-sustainable-disinfection-for-hospital-pharmacies/.

[59] Luminus Devices, Inc. *XBT-3535-265/275 nm Surface Mount UVC LED*. PDS-003329 Rev 02. Luminus Devices, Inc. Mar. 2024. URL: https://www.luminus.com.

A

Ethical Considerations

A.1. Societal impacts

A fully functional product implemented in the market will have numerous positive effects. The biggest impact will be on the environment, as the use of chemical pesticides will be heavily reduced or even eliminated. By removing pesticides from the seed treatment phase, there is no risk of chemical residues remaining on the crops after they are harvested, allowing consumers to buy produce without worrying about food contamination. Because seeds must be disinfected to optimize crop yields, this system provides farmers, seed distribution companies, and the broader agricultural sector with a method to clean seeds in an eco-friendly way. Furthermore, unlike traditional hot-water baths, this dry method avoids producing large amounts of wasteful wastewater. However, utilizing the product does come with operational dangers; UVC radiation is hazardous to human skin and especially damaging to the eyes, while the toxic ozone gas and high-voltage electricity required to generate the cold plasma add another layer of danger to this setup.

A.2. Ethical aspects of development

The system's use of UVC brings safety concerns that were carefully taken into account during the design and initial testing phases. We implemented a time delay between turning on the setup and the activation of the LEDs, giving the user enough time to get to a safe distance. In addition, the entire setup was enclosed in a protective box during both individual and integrated testing. Because UVC radiation has low penetration depth, it cannot pass through the physical structure of the box, ensuring operator safety throughout development.

Another ethical aspect arose from the component acquisition process. For this prototype, several electrical components were bought from fast-producing online marketplaces like AliExpress. These platforms can be untrustworthy and often sell products at cheap prices that may result from unethical production supply chains, lack of material traceability, or poor labor conditions.

A.3. Ethical evaluation

At the core of this project lies the value of finding a sustainable way to disinfect seeds. By reducing chemical runoff, the project aligns closely with the core bioethical principle of non-maleficence (doing no harm) toward the environment. Furthermore, it does not require enormous amounts of water. However, this environmental value must be balanced against the technical performance and agricultural safety the system guarantees.

Because using UVC as a primary seed disinfection method is relatively new, its effectiveness is not always 100% guaranteed on its own. In medical environments, UVC is rarely used in isolation; it is typically combined with other methods to achieve full cleanliness. If the system fails to fully eliminate seed-borne pathogens, it could lead to crop failure for farmers, directly affecting regional food security. This is why the combination of UVC and plasma is a crucial research area to mitigate this potential effi-

cacy gap. Additionally, testing the effectiveness of pulsed UVC versus continuous exposure is ethically valuable, as it could significantly reduce energy consumption if proven effective.

A.4. Reflection on design choices

The LEDs used in this project were chosen based on their maximum irradiation, which directly dictates the electrical current the circuit needs to supply. Because the circuit runs at these maximum ratings, it reduces the life expectancy of the LEDs and other components. During testing, certain components broke several times without a clear explanation. Reflecting on the design, a dedicated protection circuit should have been built into the setup from the beginning, which would have improved the overall durability and minimized electronic waste.

A.5. Forward-looking responsibility

To transition this prototype into a responsible real-world application, the design must be safely scaled up to handle larger bulk quantities of seeds for commercial distribution companies.

From an ethical standpoint, serious safety features must be engineered into the final commercial product to protect workers using the equipment. This should include an automated hardware and software interlock system that constantly monitors the full setup-tracking ozone levels, UV leakage, temperature, and ensuring the high-voltage supply is not malfunctioning. If the enclosure is opened or any variance is detected, a digital and manual safety lifeline must instantly shut down the entire system.

B

Source Code

B.1. Python Code for LED Configuration

```
# -*- coding: utf-8 -*-
"""LEDSimulation.ipynb

Normal␣LED␣pattern
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

tray_dia = 100  #in mm
resolution = 400  #amount of pixels

#create the tray and applying a mask to make it circular
x = np.linspace(-tray_dia/2, tray_dia/2, resolution)
y = np.linspace(-tray_dia/2, tray_dia/2, resolution)
X, Y = np.meshgrid(x, y)

distance_map = np.sqrt(X**2 + Y**2)
tray_radius = tray_dia / 2
mask = distance_map <= tray_radius

#irradiation data from badwing graph
measured_angles = np.array([0,  10,  20,  30,  35,  40,  50,  60,  70,  80,  90])
measured_int    = np.array([0.75, 0.78, 0.85, 0.95, 1.0, 0.95, 0.80, 0.60, 0.35, 0.05, 0.0])

# Linear interpolation function for the LED irradiation profile
relative_intensity = interp1d(measured_angles, measured_int, kind='linear',
                              bounds_error=False, fill_value=0.0)

#creating and going through all angles to find the relative brightness
fine_theta = np.linspace(0, np.radians(90), 1000)
sin_theta = np.sin(fine_theta)
rel_int_slice = relative_intensity(np.degrees(fine_theta))

# Compute total 3D integrated volume of the batwing profile by interating and then
    normilizing
beam_relative_volume = 2 * np.pi * np.trapz(rel_int_slice * sin_theta, fine_theta)
norm_constant = 1.0 / beam_relative_volume

def calculate_measured_irradiance(x_led, y_led, z_height, power_mw):
    #calculates irradiance using the 'relative_intensity' radiation pattern.

    #Distance from LED to each point on the grid surface
    dist_sq = (X - x_led)**2 + (Y - y_led)**2 + z_height**2
    dist = np.sqrt(dist_sq)

```

```
47      #Convert the angle of incidence (theta) in degrees
48      theta_rad = np.arccos(z_height / dist)
49      theta_deg = np.degrees(theta_rad)
50
51      #use change to degrees
52      relative_profile = relative_intensity(theta_deg)
53
54      #cosine correction factor for geometric spreading
55      cos_theta = z_height / dist
56
57      #calculate Irradiance
58      intensity = power_mw * norm_constant * relative_profile
59      irradiance = (intensity / dist_sq) * cos_theta
60      return irradiance
61
62  def get_uniformity(data_map):
63      #Calculates the coefficient of variation (CV) percentage inside the tray mask
64      avg = np.nanmean(data_map)
65      std = np.nanstd(data_map)
66      cv = (std / avg) * 100 if avg > 0 else 100.0
67      return cv, avg
68
69  def generate_symmetric_array(r1, r3):  #ad r2 for three rings
70      #Generates a  10-LED configuration over three (or two) rings.
71      # inner ring/center
72      angles_in = np.linspace(0, 2*np.pi, 1, endpoint=False)
73      inner = [(r1 * np.cos(a), r1 * np.sin(a)) for a in angles_in]
74
75      #middle ring
76      #angles_mid = np.linspace(0, 2*np.pi, 0, endpoint=False)
77      #middle = [(r2 * np.cos(a), r2 * np.sin(a)) for a in angles_mid]
78
79      # outer ring
80      angles_out = np.linspace(0, 2*np.pi, 9, endpoint=False)
81      outer = [(r3 * np.cos(a), r3 * np.sin(a)) for a in angles_out]
82
83      return inner  + outer  #ad middle for three rings
84
85  height = 30     # mm
86  P_total = 75    # mW per LED
87
88  best_error = 100.0     #set highest, replaced when lower is founnd
89  best_layout = None
90  best_map = None
91  best_radii = (0, 0) # Change to (0,0,0) for three rings
92
93  print("Optimizing layout for lowest Uniformity Error using empirical profile...")
94
95  # Systematically sweep potential radii configurations
96  for r1 in np.linspace(0, 15, 12):      # Inner ring possibilities
97    # for r2 in np.linspace(18, 30, 12): # Middle ring possibilities
98      for r3 in np.linspace(33, 48, 12): # Outer ring possibilities
99
100             #generate the 10-LED layout
101             current_locations = generate_symmetric_array(r1, r3) #this function only takes
                    two radii
102
103             #sum up total irradiance for this layout setup
104             total_irradiance = np.zeros_like(X)
105             for lx, ly in current_locations:
106                 total_irradiance += calculate_measured_irradiance(lx, ly, height, P_total)
107
108             #apply circular tray mask
109             masked_map = np.where(mask, total_irradiance, np.nan)
110
111             #evaluate how flat the light distribution is
112             error, average = get_uniformity(masked_map)
113
114             #save if layout performs better
115             if error < best_error:
116                 best_error = error
```

```
                best_layout = current_locations
                best_map = masked_map
                best_radii = (r1, r3)

print("Optimization complete!")
print(f"Best Radii Configuration found -> Inner: {best_radii[0]:.1f}mm | Outer: {best_radii[1]:.1f}mm")
print(f"Achieved Uniformity Error (CV): {best_error:.2f}%")

# Plotting
plt.figure(figsize=(7, 6))
plt.pcolormesh(X, Y, best_map, shading='auto', cmap='magma')
plt.colorbar(label='Irradiance (mW/mm²)')

lx_coords = [loc[0] for loc in best_layout]
ly_coords = [loc[1] for loc in best_layout]
plt.scatter(lx_coords, ly_coords, color='white', marker='s', s=40, edgecolors='black', label='LEDs')

plt.title(f"Array Simulation: {len(best_layout)} LEDs at {height}mm Height")
plt.xlabel("x [mm]")
plt.ylabel("y [mm]")
plt.legend()
plt.axis('equal')
plt.show()

"""Tray+circular"""

import numpy as np
import matplotlib.pyplot as plt
import random
from scipy.optimize import minimize
from scipy.interpolate import interp1d

manual_coordinates = [ #adding the selected coordinated by hand, can be replaced with optimizing code
    (0.00, 0.00),
    (48.00,   0.00),   # 0 degrees
    (36.77,  30.85),   # 40 degrees
    (8.34,   47.27),   # 80 degrees
    (-24.00,  41.57),  # 120 degrees
    (-45.11,  16.42),  # 160 degrees
    (-45.11, -16.42),  # 200 degrees
    (-24.00, -41.57),  # 240 degrees
    (8.34,  -47.27),  # 280 degrees
    (36.77,  -30.85)   # 320 degrees
]

tray_dia = 100
resolution = 400

#create the tray and applying a mask to make it circular
x = np.linspace(-tray_dia/2, tray_dia/2, resolution)
y = np.linspace(-tray_dia/2, tray_dia/2, resolution)

X, Y = np.meshgrid(x, y)

distance_map = np.sqrt(X**2 + Y**2)
tray_radius = tray_dia/2
tray_mask = distance_map <= tray_radius

P_total = 75.0       # mW per LED
led_height = 30.0    # mm
obstacle_height = 5.0  #mm height of plasme panel

#defining the dimentions of the plasma tray
rect_x_min, rect_x_max = -65, 65
rect_y_min, rect_y_max = -40, 40

# 10mm border
grid_x_min, grid_x_max = rect_x_min + 10, rect_x_max - 10
```

```
grid_y_min, grid_y_max = rect_y_min + 10, rect_y_max - 10

hole_size = 4.0   #mm  space between cappilaries
wall_size = 0.1   #mm  thickness cappilaries
period = hole_size + wall_size

#create all the individual lines
x_walls = np.arange(grid_x_min, grid_x_max + 0.1, period)
y_walls = np.arange(grid_y_min, grid_y_max + 0.1, period)

#irradiation data from badwing graph
measured_angles = np.array([0,  10,  20,  30,  35,  40,  50,  60,  70,  80,  90])
measured_int    = np.array([0.75, 0.78, 0.85, 0.95, 1.0, 0.95, 0.80, 0.60, 0.35, 0.05, 0.0])

# Linear interpolation function for the LED irradiation profile
relative_intensity = interp1d(measured_angles, measured_int, kind='linear',
                              bounds_error=False, fill_value=0.0)

#creating and going through all angles to find the relative brightness
fine_theta = np.linspace(0, np.radians(90), 1000)
sin_theta = np.sin(fine_theta)
rel_int_slice = relative_intensity(np.degrees(fine_theta))

# Compute total 3D integrated volume of the batwing profile by interating and then
    normilizing
beam_relative_volume = 2 * np.pi * np.trapz(rel_int_slice * sin_theta, fine_theta)
norm_constant = 1.0 / beam_relative_volume

def get_uniformity(data_map):
    #Calculates the coefficient of variation (CV) percentage inside the tray mask
    valid_data = data_map[tray_mask]
    avg = np.nanmean(valid_data)
    std = np.nanstd(valid_data)
    cv = (std / avg) * 100  if avg > 0 else 100.0
    return cv, avg

def calculate_attenuated_irradiance(layout, z_led, z_obs, power_mw):
    total_irradiance = np.zeros_like(X)
    shadow_ratio = z_obs / z_led

    #Distance from LED to each point on the grid surface
    for x_led, y_led in layout:
        dist_sq = (X - x_led)**2 + (Y - y_led)**2 + z_led**2
        dist = np.sqrt(dist_sq)

        #Convert the angle of incidence (theta) in degrees
        theta_rad = np.arccos(z_led / dist)
        theta_deg = np.degrees(theta_rad)
        relative_profile = relative_intensity(theta_deg)
        cos_theta = z_led / dist

        #norm_constant scaling
        intensity = power_mw * norm_constant * relative_profile
        E_raw = (intensity / dist_sq) * cos_theta

        #project coordinates back to the obstacle grid height
        X_obs = X - shadow_ratio * (X - x_led)
        Y_obs = Y - shadow_ratio * (Y - y_led)

        #frame check, if there is a frame radiance cannot pass through
        in_frame = (X_obs >= rect_x_min) & (X_obs <= rect_x_max) & \
                   (Y_obs >= rect_y_min) & (Y_obs <= rect_y_max)

        in_clearing = (X_obs > grid_x_min) & (X_obs < grid_x_max) & \
                      (Y_obs > grid_y_min) & (Y_obs < grid_y_max)

        is_solid_border = in_frame & ~in_clearing

        #grid wall check
        local_x = (X_obs - grid_x_min) % period
        local_y = (Y_obs - grid_y_min) % period
```

```
        on_grid_wall = (local_x >= hole_size) | (local_y >= hole_size)

        blocked = is_solid_border | (in_clearing & on_grid_wall)
        total_irradiance += np.where(blocked, 0.0, E_raw)

    return total_irradiance

def simulate_rotation(static_map):
    #extracting the needed static information
    r_pixel = distance_map[tray_mask]
    val_pixel = static_map[tray_mask]

    #creating individual rings, whos iradiance we will find
    bin_edges = np.linspace(0, tray_radius, 80)
    bin_centers = 0.5 * (bin_edges[:-1] + bin_edges[1:])

    #Finding the total irradiance per ring
    radial_profile, _ = np.histogram(r_pixel, bins=bin_edges, weights=val_pixel)
    counts, _ = np.histogram(r_pixel, bins=bin_edges)

    #Finding the average
    radial_profile = np.divide(radial_profile, counts, out=np.zeros_like(radial_profile),
        where=counts > 0)
    r_to_intensity = interp1d(bin_centers, radial_profile, kind='linear', bounds_error=False,
         fill_value="extrapolate")

    return np.where(tray_mask, r_to_intensity(distance_map), np.nan)

print(f"Loaded␣{len(manual_coordinates)}␣manual␣LED␣coordinates.")

#static figure and rotational figure
final_static = calculate_attenuated_irradiance(manual_coordinates, led_height,
    obstacle_height, P_total)
final_rotated = simulate_rotation(final_static)

final_cv, final_avg_irradiance = get_uniformity(final_rotated)

print(f"Rotational␣Uniformity␣(CV):␣{final_cv:.2f}%")
print(f"Average␣Irradiance:␣{final_avg_irradiance:.4f}␣mW/mm²␣(Equals␣{final_avg_irradiance␣*
    ␣100:.2f}␣mW/cm²)")

exposure_time_sec = 10 * 60
final_dosage_map = final_rotated * exposure_time_sec

#plotting
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6.5))

im1 = ax1.pcolormesh(X, Y, np.where(tray_mask, final_static, np.nan), shading='auto', cmap='
    magma')
ax1.scatter([l[0] for l in manual_coordinates], [l[1] for l in manual_coordinates], color='
    cyan', marker='o', s=60, edgecolors='black', label='LEDs')
ax1.set_title("Symmetric␣Optimal␣Static␣Snapshot")
ax1.axis('equal')
fig.colorbar(im1, ax=ax1, label='Irradiance␣(mW/mm²)')

im2 = ax2.pcolormesh(X, Y, final_dosage_map, shading='auto', cmap='viridis')
ax2.set_title(f"10-Min␣Rotational␣Dose\nCV:␣{final_cv:.2f}%␣|␣Avg:␣{final_avg_irradiance␣*␣
    exposure_time_sec:.1f}␣mJ/mm²")
ax2.axis('equal')
fig.colorbar(im2, ax=ax2, label='Total␣Dose␣(mJ/mm²)')

plt.suptitle("Globally␣Optimized␣10-LED␣Array␣(10cm␣Tray␣at␣35mm␣Height)", fontsize=14,
    weight='bold')
plt.show()
```

B.2. C Code Microcontroller

The provided code sections mostly represent individual hardware components that were developed and tested independently, each in its own main.c file. For the final product, all functionality must be integrated into a single unified codebase with the display interface, requiring code reorganization across

initialization routines and the main loop, potential variable renaming to avoid conflicts, reconciliation of shared peripherals like timers, GPIO pins, and interrupts, coordination of timing and task scheduling to prevent performance issues, and minor functional adjustments to ensure all components operate harmoniously together.

B.2.1. PWM

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2026 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */
volatile uint32_t pulse_width_us = 300; //pulse width in us
volatile uint32_t pwm_frequency_hz = 1000; //frequency

volatile uint32_t run_duration_seconds = 0; //how long device will be on
volatile uint32_t pwm_start_tick = 0; //at what time system clock tick the pwm was started
volatile uint8_t  pwm_running = 0; //0 is stopped, 1 is running; flag to check if uv is on


volatile uint8_t  duty_cycle = 30;



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```
64 static void MX_GPIO_Init(void);
65 static void MX_TIM1_Init(void);
66 static void MX_TIM4_Init(void);
67 static void MX_TIM2_Init(void);
68 /* USER CODE BEGIN PFP */
69
70 void start_pwm_with_timer(uint32_t duration_seconds, uint32_t delay_ms);
71 void stop_pwm(void);
72 void set_pulse_width_us(uint32_t width_us, uint32_t frequency_hz);
73 uint32_t calculate_ARR(uint32_t frequency_hz);
74
75 /* USER CODE END PFP */
76
77 /* Private user code ---------------------------------------------------------*/
78 /* USER CODE BEGIN 0 */
79
80 /* USER CODE END 0 */
81
82 /**
83   * @brief  The application entry point.
84   * @retval int
85   */
86 int main(void)
87 {
88
89   /* USER CODE BEGIN 1 */
90
91   /* USER CODE END 1 */
92
93   /* MCU Configuration--------------------------------------------------------*/
94
95   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
96   HAL_Init();
97
98   /* USER CODE BEGIN Init */
99
100  /* USER CODE END Init */
101
102  /* Configure the system clock */
103  SystemClock_Config();
104
105  /* USER CODE BEGIN SysInit */
106
107  /* USER CODE END SysInit */
108
109  /* Initialize all configured peripherals */
110  MX_GPIO_Init();
111  MX_TIM1_Init();
112  MX_TIM4_Init();
113  MX_TIM2_Init();
114  /* USER CODE BEGIN 2 */
115
116  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
117
118
119  set_pulse_width_us(300, pwm_frequency_hz);
120  start_pwm_with_timer(60*10, 1000*15); //seconds on-duration, milliseconds pause before
       start
121
122  /* USER CODE END 2 */
123
124  /* Infinite loop */
125  /* USER CODE BEGIN WHILE */
126  while (1)
127  {
128    /* USER CODE END WHILE */
129
130    /* USER CODE BEGIN 3 */
131                if (pwm_running && run_duration_seconds > 0) { //if pwm flag is on and the
                       run duration set isnt infinite
```

```
132                         if (HAL_GetTick() - pwm_start_tick >= run_duration_seconds * 1000) {
                                //hal tick measures microseconds since boot
133                             stop_pwm();
134                         }
135                 }
136
137                 HAL_Delay(10); //loop at 100 Hz
138
139   }
140   /* USER CODE END 3 */
141 }
142
143 /**
144   * @brief System Clock Configuration
145   * @retval None
146   */
147 void SystemClock_Config(void)
148 {
149   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
150   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
151
152   /** Configure the main internal regulator output voltage
153   */
154   __HAL_RCC_PWR_CLK_ENABLE();
155   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
156
157   /** Initializes the RCC Oscillators according to the specified parameters
158   * in the RCC_OscInitTypeDef structure.
159   */
160   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
161   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
162   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
163   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
164   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
165   RCC_OscInitStruct.PLL.PLLM = 8;
166   RCC_OscInitStruct.PLL.PLLN = 100;
167   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
168   RCC_OscInitStruct.PLL.PLLQ = 4;
169   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
170   {
171     Error_Handler();
172   }
173
174   /** Initializes the CPU, AHB and APB buses clocks
175   */
176   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
177                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
178   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
179   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
180   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
181   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
182
183   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
184   {
185     Error_Handler();
186   }
187 }
188
189 /**
190   * @brief TIM1 Initialization Function
191   * @param None
192   * @retval None
193   */
194 static void MX_TIM1_Init(void)
195 {
196
197   /* USER CODE BEGIN TIM1_Init 0 */
198
199   /* USER CODE END TIM1_Init 0 */
200
201   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
```

```
202   TIM_MasterConfigTypeDef sMasterConfig = {0};
203   TIM_OC_InitTypeDef sConfigOC = {0};
204   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
205
206   /* USER CODE BEGIN TIM1_Init 1 */
207
208   /* USER CODE END TIM1_Init 1 */
209   htim1.Instance = TIM1;
210   htim1.Init.Prescaler = 99;
211   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
212   htim1.Init.Period = 999;
213   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
214   htim1.Init.RepetitionCounter = 0;
215   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
216   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
217   {
218     Error_Handler();
219   }
220   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
221   if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
222   {
223     Error_Handler();
224   }
225   if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
226   {
227     Error_Handler();
228   }
229   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
230   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
231   if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
232   {
233     Error_Handler();
234   }
235   sConfigOC.OCMode = TIM_OCMODE_PWM1;
236   sConfigOC.Pulse = 100;
237   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
238   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
239   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
240   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
241   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
242   if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
243   {
244     Error_Handler();
245   }
246   sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
247   sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
248   sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
249   sBreakDeadTimeConfig.DeadTime = 0;
250   sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
251   sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
252   sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
253   if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
254   {
255     Error_Handler();
256   }
257   /* USER CODE BEGIN TIM1_Init 2 */
258
259   /* USER CODE END TIM1_Init 2 */
260   HAL_TIM_MspPostInit(&htim1);
261
262 }
263
264 /**
265   * @brief TIM2 Initialization Function
266   * @param None
267   * @retval None
268   */
269 static void MX_TIM2_Init(void)
270 {
271
272   /* USER CODE BEGIN TIM2_Init 0 */
```

```
273
274   /* USER CODE END TIM2_Init 0 */
275
276   TIM_MasterConfigTypeDef sMasterConfig = {0};
277   TIM_OC_InitTypeDef sConfigOC = {0};
278
279   /* USER CODE BEGIN TIM2_Init 1 */
280
281   /* USER CODE END TIM2_Init 1 */
282   htim2.Instance = TIM2;
283   htim2.Init.Prescaler = 99;
284   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
285   htim2.Init.Period = 999;
286   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
287   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
288   if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
289   {
290     Error_Handler();
291   }
292   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
293   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
294   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
295   {
296     Error_Handler();
297   }
298   sConfigOC.OCMode = TIM_OCMODE_PWM1;
299   sConfigOC.Pulse = 300;
300   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
301   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
302   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
303   {
304     Error_Handler();
305   }
306   /* USER CODE BEGIN TIM2_Init 2 */
307
308   /* USER CODE END TIM2_Init 2 */
309   HAL_TIM_MspPostInit(&htim2);
310
311 }
312
313 /**
314   * @brief TIM4 Initialization Function
315   * @param None
316   * @retval None
317   */
318 static void MX_TIM4_Init(void)
319 {
320
321   /* USER CODE BEGIN TIM4_Init 0 */
322
323   /* USER CODE END TIM4_Init 0 */
324
325   TIM_MasterConfigTypeDef sMasterConfig = {0};
326   TIM_OC_InitTypeDef sConfigOC = {0};
327
328   /* USER CODE BEGIN TIM4_Init 1 */
329
330   /* USER CODE END TIM4_Init 1 */
331   htim4.Instance = TIM4;
332   htim4.Init.Prescaler = 99;
333   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
334   htim4.Init.Period = 999;
335   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
336   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
337   if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
338   {
339     Error_Handler();
340   }
341   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
342   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
343   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
```

```
344   {
345     Error_Handler();
346   }
347   sConfigOC.OCMode = TIM_OCMODE_PWM1;
348   sConfigOC.Pulse = 300;
349   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
350   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
351   if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
352   {
353     Error_Handler();
354   }
355   /* USER CODE BEGIN TIM4_Init 2 */
356
357   /* USER CODE END TIM4_Init 2 */
358   HAL_TIM_MspPostInit(&htim4);
359
360 }
361
362 /**
363   * @brief GPIO Initialization Function
364   * @param None
365   * @retval None
366   */
367 static void MX_GPIO_Init(void)
368 {
369   GPIO_InitTypeDef GPIO_InitStruct = {0};
370   /* USER CODE BEGIN MX_GPIO_Init_1 */
371
372   /* USER CODE END MX_GPIO_Init_1 */
373
374   /* GPIO Ports Clock Enable */
375   __HAL_RCC_GPIOC_CLK_ENABLE();
376   __HAL_RCC_GPIOA_CLK_ENABLE();
377   __HAL_RCC_GPIOB_CLK_ENABLE();
378
379   /*Configure GPIO pin Output Level */
380   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
381
382   /*Configure GPIO pin : LED_Pin */
383   GPIO_InitStruct.Pin = LED_Pin;
384   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
385   GPIO_InitStruct.Pull = GPIO_NOPULL;
386   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
387   HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);
388
389   /* USER CODE BEGIN MX_GPIO_Init_2 */
390
391   /* USER CODE END MX_GPIO_Init_2 */
392 }
393
394 /* USER CODE BEGIN 4 */
395
396
397
398 // duration_seconds: how long to run (0 = run forever)
399 // delay_ms: wait before starting (for startup delay)
400 void start_pwm_with_timer(uint32_t duration_seconds, uint32_t delay_ms) {
401         run_duration_seconds = duration_seconds;
402
403         if(delay_ms>0){ //this is for pre-start delay
404                 HAL_Delay(delay_ms);
405         }
406         // start timer 1 pwm on PA8 (output to gate driver)
407         HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); //starts pwm generation
408         __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pulse_width_us); //changes duty cycle
              through CCR register
409
410         HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1); //starts pwm generation
411         __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, pulse_width_us); //changes duty cycle
              through CCR register
412
```

```
413         HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //starts pwm generation
414         __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pulse_width_us); //changes duty cycle
                through CCR register
415
416         // turn on LED when UV is on
417         HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
418
419         //record start time (tick) and set flag
420         pwm_start_tick = HAL_GetTick();
421         pwm_running = 1;
422
423 }
424
425 // stop PWM and turn off output
426 void stop_pwm(void) {
427         HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1); //stop TIM1 PWM generation
428         HAL_TIM_PWM_Stop(&htim4, TIM_CHANNEL_1);
429         HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1);
430
431         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);  // force output to gate driver
                low
432         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);  // force output to gate driver
                low
433         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);  // force output to gate driver
                low
434
435         HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); //force output to on-
                indicator LED low
436         pwm_running = 0; //set running flag to 0 (off)
437 }
438
439 //calculates required ARR to get a specific frequency
440 // timer clock = 1,000,000 Hz (1 us per tick)
441 // ARR = (clock / DesiredFrequency) - 1
442 uint32_t calculate_ARR(uint32_t frequency_hz) { //ARR = auto-reload register -> decides
        frequency by resetting counter at specific tick (depends on clk speed)
443     return (1000000 / frequency_hz) - 1;
444 }
445
446
447 //For display
448 // set the pulse width in microseconds
449 void set_pulse_width_us(uint32_t width_us, uint32_t frequency_hz) {
450     uint32_t period_us = 1000000 / frequency_hz;
451
452         if (width_us < 10)  width_us = 10;   // Minimum 10 µs (1% duty cycle of smallest freq
                : 1 kHz)
453     if (width_us > period_us) width_us = period_us;
454
455
456     pulse_width_us = width_us;
457
458     if (pwm_running) {
459         __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pulse_width_us);
460         __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, pulse_width_us);
461         __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pulse_width_us);
462
463
464     }
465 }
466
467 void set_pwm_frequency(uint32_t frequency_hz) {
468     uint32_t arr = calculate_ARR(frequency_hz);
469     __HAL_TIM_SET_AUTORELOAD(&htim1, arr);
470     __HAL_TIM_SET_AUTORELOAD(&htim4, arr);
471     pwm_frequency_hz = frequency_hz;
472 }
473 /* USER CODE END 4 */
474
475 /**
476   * @brief  This function is executed in case of error occurrence.
```

```
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

B.2.2. Rotary Encoder

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2026 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

```

```
40 /* USER CODE END PM */
41
42 /* Private variables ---------------------------------------------------------*/
43 TIM_HandleTypeDef htim4;
44
45 /* USER CODE BEGIN PV */
46
47 /* USER CODE END PV */
48
49 /* Private function prototypes -----------------------------------------------*/
50 void SystemClock_Config(void);
51 static void MX_GPIO_Init(void);
52 static void MX_TIM4_Init(void);
53 /* USER CODE BEGIN PFP */
54
55 /* USER CODE END PFP */
56
57 /* Private user code ---------------------------------------------------------*/
58 /* USER CODE BEGIN 0 */
59 /* USER CODE END 0 */
60
61 /**
62   * @brief  The application entry point.
63   * @retval int
64   */
65 int main(void)
66 {
67
68   /* USER CODE BEGIN 1 */
69
70   /* USER CODE END 1 */
71
72   /* MCU Configuration--------------------------------------------------------*/
73
74   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
75   HAL_Init();
76
77   /* USER CODE BEGIN Init */
78
79   /* USER CODE END Init */
80
81   /* Configure the system clock */
82   SystemClock_Config();
83
84   /* USER CODE BEGIN SysInit */
85
86   /* USER CODE END SysInit */
87
88   /* Initialize all configured peripherals */
89   MX_GPIO_Init();
90   MX_TIM4_Init();
91   /* USER CODE BEGIN 2 */
92   HAL_TIM_Encoder_Start(&htim4,TIM_CHANNEL_ALL);
93   int16_t previous_enc_count = (int16_t) TIM4->CNT;
94   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
95   int16_t current_enc_count = (int16_t) TIM4->CNT;
96   /* USER CODE END 2 */
97
98   /* Infinite loop */
99   /* USER CODE BEGIN WHILE */
100  while (1)
101  {
102    /* USER CODE END WHILE */
103
104    /* USER CODE BEGIN 3 */
105
106          //turning:
107          current_enc_count = (int16_t) TIM4->CNT;
108          //enc_difference = previous_enc_count - current_enc_count;
109
110          if(current_enc_count < previous_enc_count){
```

```
111                     //turned to the right by +enc_difference
112
113
114             }
115             else if(current_enc_count > previous_enc_count){
116                     //turned to the left by -enc_difference
117
118             previous_enc_count = current_enc_count;
119 //          return enc_difference;
120             //clicking:
121 //          if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)==GPIO_PIN_RESET){
122 //clicked
123                     HAL_Delay(200); //debounce 200 ms
124 //          }
125   }
126   /* USER CODE END 3 */
127 }
128
129 /**
130   * @brief System Clock Configuration
131   * @retval None
132   */
133 void SystemClock_Config(void)
134 {
135   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
136   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
137
138   /** Configure the main internal regulator output voltage
139   */
140   __HAL_RCC_PWR_CLK_ENABLE();
141   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
142
143   /** Initializes the RCC Oscillators according to the specified parameters
144   * in the RCC_OscInitTypeDef structure.
145   */
146   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
147   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
148   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
149   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
150   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
151   {
152     Error_Handler();
153   }
154
155   /** Initializes the CPU, AHB and APB buses clocks
156   */
157   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
158                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
159   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
160   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
161   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
162   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
163
164   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
165   {
166     Error_Handler();
167   }
168 }
169
170 /**
171   * @brief TIM4 Initialization Function
172   * @param None
173   * @retval None
174   */
175 static void MX_TIM4_Init(void)
176 {
177
178   /* USER CODE BEGIN TIM4_Init 0 */
179
180   /* USER CODE END TIM4_Init 0 */
181
```

```
182   TIM_Encoder_InitTypeDef sConfig = {0};
183   TIM_MasterConfigTypeDef sMasterConfig = {0};
184
185   /* USER CODE BEGIN TIM4_Init 1 */
186
187   /* USER CODE END TIM4_Init 1 */
188   htim4.Instance = TIM4;
189   htim4.Init.Prescaler = 0;
190   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
191   htim4.Init.Period = 65535;
192   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
193   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
194   sConfig.EncoderMode = TIM_ENCODERMODE_TI1;
195   sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
196   sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
197   sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
198   sConfig.IC1Filter = 5;
199   sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
200   sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
201   sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
202   sConfig.IC2Filter = 5;
203   if (HAL_TIM_Encoder_Init(&htim4, &sConfig) != HAL_OK)
204   {
205     Error_Handler();
206   }
207   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
208   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
209   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
210   {
211     Error_Handler();
212   }
213   /* USER CODE BEGIN TIM4_Init 2 */
214
215   /* USER CODE END TIM4_Init 2 */
216
217 }
218
219 /**
220   * @brief GPIO Initialization Function
221   * @param None
222   * @retval None
223   */
224 static void MX_GPIO_Init(void)
225 {
226   GPIO_InitTypeDef GPIO_InitStruct = {0};
227   /* USER CODE BEGIN MX_GPIO_Init_1 */
228
229   /* USER CODE END MX_GPIO_Init_1 */
230
231   /* GPIO Ports Clock Enable */
232   __HAL_RCC_GPIOC_CLK_ENABLE();
233   __HAL_RCC_GPIOB_CLK_ENABLE();
234
235   /*Configure GPIO pin Output Level */
236   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
237
238   /*Configure GPIO pin : LED_Pin */
239   GPIO_InitStruct.Pin = LED_Pin;
240   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
241   GPIO_InitStruct.Pull = GPIO_NOPULL;
242   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
243   HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);
244
245   /*Configure GPIO pin : ENC_KEY_Pin */
246   GPIO_InitStruct.Pin = ENC_KEY_Pin;
247   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
248   GPIO_InitStruct.Pull = GPIO_PULLUP;
249   HAL_GPIO_Init(ENC_KEY_GPIO_Port, &GPIO_InitStruct);
250
251   /* USER CODE BEGIN MX_GPIO_Init_2 */
252
```

```
253   /* USER CODE END MX_GPIO_Init_2 */
254 }
255
256 /* USER CODE BEGIN 4 */
257
258 /* USER CODE END 4 */
259
260 /**
261   * @brief  This function is executed in case of error occurrence.
262   * @retval None
263   */
264 void Error_Handler(void)
265 {
266   /* USER CODE BEGIN Error_Handler_Debug */
267   /* User can add his own implementation to report the HAL error return state */
268   __disable_irq();
269   while (1)
270   {
271           HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
272
273   }
274   /* USER CODE END Error_Handler_Debug */
275 }
276 #ifdef USE_FULL_ASSERT
277 /**
278   * @brief  Reports the name of the source file and the source line number
279   *         where the assert_param error has occurred.
280   * @param  file: pointer to the source file name
281   * @param  line: assert_param error line source number
282   * @retval None
283   */
284 void assert_failed(uint8_t *file, uint32_t line)
285 {
286   /* USER CODE BEGIN 6 */
287   /* User can add his own implementation to report the file name and line number,
288      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
289   /* USER CODE END 6 */
290 }
291 #endif /* USE_FULL_ASSERT */
```

B.2.3. Ozone Sensor

```
1  /* USER CODE BEGIN Header */
2  /**
3    ******************************************************************************
4    * @file           : main.c
5    * @brief          : Main program body
6    ******************************************************************************
7    * @attention
8    *
9    * Copyright (c) 2026 STMicroelectronics.
10   * All rights reserved.
11   *
12   * This software is licensed under terms that can be found in the LICENSE file
13   * in the root directory of this software component.
14   * If no LICENSE file comes with this software, it is provided AS-IS.
15   *
16   ******************************************************************************
17   */
18 /* USER CODE END Header */
19 /* Includes ------------------------------------------------------------------*/
20 #include "main.h"
21
22 /* Private includes ----------------------------------------------------------*/
23 /* USER CODE BEGIN Includes */
24
25 /* USER CODE END Includes */
26
27 /* Private typedef -----------------------------------------------------------*/
28 /* USER CODE BEGIN PTD */
```

```
29
30 /* USER CODE END PTD */
31
32 /* Private define ------------------------------------------------------------*/
33 /* USER CODE BEGIN PD */
34
35 /* USER CODE END PD */
36
37 /* Private macro -------------------------------------------------------------*/
38 /* USER CODE BEGIN PM */
39
40 /* USER CODE END PM */
41
42 /* Private variables ---------------------------------------------------------*/
43 ADC_HandleTypeDef hadc1;
44
45 /* USER CODE BEGIN PV */
46
47 /* USER CODE END PV */
48
49 /* Private function prototypes -----------------------------------------------*/
50 void SystemClock_Config(void);
51 static void MX_GPIO_Init(void);
52 static void MX_ADC1_Init(void);
53 /* USER CODE BEGIN PFP */
54
55 /* USER CODE END PFP */
56
57 /* Private user code ---------------------------------------------------------*/
58 /* USER CODE BEGIN 0 */
59
60 /* USER CODE END 0 */
61
62 /**
63   * @brief  The application entry point.
64   * @retval int
65   */
66 int main(void)
67 {
68
69   /* USER CODE BEGIN 1 */
70
71   /* USER CODE END 1 */
72
73   /* MCU Configuration--------------------------------------------------------*/
74
75   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
76   HAL_Init();
77
78   /* USER CODE BEGIN Init */
79
80   /* USER CODE END Init */
81
82   /* Configure the system clock */
83   SystemClock_Config();
84
85   /* USER CODE BEGIN SysInit */
86
87   /* USER CODE END SysInit */
88
89   /* Initialize all configured peripherals */
90   MX_GPIO_Init();
91   MX_ADC1_Init();
92   /* USER CODE BEGIN 2 */
93
94   /* USER CODE END 2 */
95
96   /* Infinite loop */
97   /* USER CODE BEGIN WHILE */
98   while (1)
99   {
```

```
100     /* USER CODE END WHILE */
101
102     /* USER CODE BEGIN 3 */
103   }
104   /* USER CODE END 3 */
105 }
106
107 /**
108   * @brief System Clock Configuration
109   * @retval None
110   */
111 void SystemClock_Config(void)
112 {
113   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
114   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
115
116   /** Configure the main internal regulator output voltage
117   */
118   __HAL_RCC_PWR_CLK_ENABLE();
119   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
120
121   /** Initializes the RCC Oscillators according to the specified parameters
122   * in the RCC_OscInitTypeDef structure.
123   */
124   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
125   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
126   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
127   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
128   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
129   RCC_OscInitStruct.PLL.PLLM = 16;
130   RCC_OscInitStruct.PLL.PLLN = 192;
131   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
132   RCC_OscInitStruct.PLL.PLLQ = 4;
133   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
134   {
135     Error_Handler();
136   }
137
138   /** Initializes the CPU, AHB and APB buses clocks
139   */
140   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
141                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
142   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
143   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
144   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
145   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
146
147   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
148   {
149     Error_Handler();
150   }
151 }
152
153 /**
154   * @brief ADC1 Initialization Function
155   * @param None
156   * @retval None
157   */
158 static void MX_ADC1_Init(void)
159 {
160
161   /* USER CODE BEGIN ADC1_Init 0 */
162
163   /* USER CODE END ADC1_Init 0 */
164
165   ADC_ChannelConfTypeDef sConfig = {0};
166
167   /* USER CODE BEGIN ADC1_Init 1 */
168
169   /* USER CODE END ADC1_Init 1 */
170
```

```
171   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number
          of conversion)
172   */
173   hadc1.Instance = ADC1;
174   hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
175   hadc1.Init.Resolution = ADC_RESOLUTION_12B;
176   hadc1.Init.ScanConvMode = DISABLE;
177   hadc1.Init.ContinuousConvMode = DISABLE;
178   hadc1.Init.DiscontinuousConvMode = DISABLE;
179   hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
180   hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
181   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
182   hadc1.Init.NbrOfConversion = 1;
183   hadc1.Init.DMAContinuousRequests = DISABLE;
184   hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
185   if (HAL_ADC_Init(&hadc1) != HAL_OK)
186   {
187     Error_Handler();
188   }
189
190   /** Configure for the selected ADC regular channel its corresponding rank in the sequencer
          and its sample time.
191   */
192   sConfig.Channel = ADC_CHANNEL_1;
193   sConfig.Rank = 1;
194   sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
195   if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
196   {
197     Error_Handler();
198   }
199   /* USER CODE BEGIN ADC1_Init 2 */
200
201   /* USER CODE END ADC1_Init 2 */
202
203 }
204
205 /**
206   * @brief GPIO Initialization Function
207   * @param None
208   * @retval None
209   */
210 static void MX_GPIO_Init(void)
211 {
212   /* USER CODE BEGIN MX_GPIO_Init_1 */
213
214   /* USER CODE END MX_GPIO_Init_1 */
215
216   /* GPIO Ports Clock Enable */
217   __HAL_RCC_GPIOA_CLK_ENABLE();
218
219   /* USER CODE BEGIN MX_GPIO_Init_2 */
220
221   /* USER CODE END MX_GPIO_Init_2 */
222 }
223
224 /* USER CODE BEGIN 4 */
225
226 //FOR THE OZONE SENSOR READING:
227 float MQ131_PPB(void) //return a float
228 {
229     uint32_t adc_sum = 0;
230     HAL_ADC_Start(&hadc1); //turn on ADC hardware
231     for (int i = 0; i < 10; i++) //take average of 10 readings to average out noise
232     {
233         HAL_ADC_PollForConversion(&hadc1, 100); //poll & after 100ms timeout
234         adc_sum += HAL_ADC_GetValue(&hadc1); //read from sensor
235     }
236     HAL_ADC_Stop(&hadc1); //turn off ADC hardware
237
238
239     //200 ppb = 1V according to datasheet
```

```
240     //1000 ppb = 5v = max values --> linear connection
241     // 1v/200ppb
242     float adc_avg = (float) adc_sum /10.0;
243     float voltage_at_pin = (adc_avg * 3.3)/4095.0;     //12bitADC=4095;ADC input is capped at
            3.3V with voltage divider resistor so multiply it back to actual value
244     float sensor_voltage = voltage_at_pin*(5.0/3.3);    // reverse effects of voltage divider
            turns 5v into 3.3
245     float ppb = sensor_voltage *200.0;              // 200 ppb/volt
246
247     if (ppb<0){
248         ppb = 0;
249     }
250     if (ppb>1000){
251         ppb = 1000;
252     }
253
254     return ppb;
255 }
256
257 /* USER CODE END 4 */
258
259 /**
260   * @brief  This function is executed in case of error occurrence.
261   * @retval None
262   */
263 void Error_Handler(void)
264 {
265   /* USER CODE BEGIN Error_Handler_Debug */
266   /* User can add his own implementation to report the HAL error return state */
267   __disable_irq();
268   while (1)
269   {
270   }
271   /* USER CODE END Error_Handler_Debug */
272 }
273 #ifdef USE_FULL_ASSERT
274 /**
275   * @brief  Reports the name of the source file and the source line number
276   *         where the assert_param error has occurred.
277   * @param  file: pointer to the source file name
278   * @param  line: assert_param error line source number
279   * @retval None
280   */
281 void assert_failed(uint8_t *file, uint32_t line)
282 {
283   /* USER CODE BEGIN 6 */
284   /* User can add his own implementation to report the file name and line number,
285      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
286   /* USER CODE END 6 */
287 }
288 #endif /* USE_FULL_ASSERT */
```

B.2.4. Display

```
1     /* USER CODE BEGIN Header */
2 /**
3  ******************************************************************************
4  * @file           : main.c
5  * @brief          : Main program body
6  ******************************************************************************
7  * @attention
8  *
9  * Copyright (c) 2026 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 ******************************************************************************
```

```
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "i2c.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ssd1306.h"
#include "ssd1306_fonts.h"
#include <stdio.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
uint32_t ADC_Read_Channel(uint32_t channel);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {

        /* USER CODE BEGIN 1 */

        /* USER CODE END 1 */

        /* MCU Configuration--------------------------------------------------------*/

        /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
        HAL_Init();

        /* USER CODE BEGIN Init */

        /* USER CODE END Init */

        /* Configure the system clock */
        SystemClock_Config();

        /* USER CODE BEGIN SysInit */

```

```
88          /* USER CODE END SysInit */
89
90          /* Initialize all configured peripherals */
91          MX_GPIO_Init();
92          MX_ADC1_Init();
93          MX_I2C1_Init();
94          /* USER CODE BEGIN 2 */
95          HAL_Delay(100); // Give the display power rail a tiny split second to stabilize
96          ssd1306_Init();
97          ssd1306_Fill(Black);
98          ssd1306_UpdateScreen();
99          /* USER CODE END 2 */
100
101         /* Infinite loop */
102         /* USER CODE BEGIN WHILE */
103         while (1) {
104                 // READ UV SENSOR (Analog PA0 -> ADC1_CHANNEL_1 on STM32G4)
105                 uint32_t raw_uv = ADC_Read_Channel(ADC_CHANNEL_1);
106                 float uv_voltage = (raw_uv * 3.3f) / 4095.0f;
107                 float uv_index = uv_voltage / 0.1f;
108
109                 //READ HDC2080 TEMP & HUMIDITY (I2C 0x40)
110
111                 uint8_t start_reg = 0x00; // Declared cleanly outside the conditional
                        statement
112                 uint8_t data[4] = { 0 };
113                 float temperature = 0.0f;
114                 float humidity = 0.0f;
115
116                 // 1. Transmit register location to slave address
117                 if (HAL_I2C_Master_Transmit(&hi2c1, (0x40 << 1), &start_reg, 1, 100)
118                                 == HAL_OK) {
119                         // 2. Receive the 4 sequential bytes back
120                         if (HAL_I2C_Master_Receive(&hi2c1, (0x40 << 1), data, 4, 100)
121                                         == HAL_OK) {
122                                 // Reconstruct 16-bit register words
123                                 uint16_t raw_temp = (data[1] << 8) | data[0];
124                                 uint16_t raw_humid = (data[3] << 8) | data[2];
125
126                                 // Apply Texas Instruments datasheet conversion curves
127                                 temperature = ((float) raw_temp / 65536.0f) * 165.0f - 40.0f;
128                                 humidity = ((float) raw_humid / 65536.0f) * 100.0f;
129                         }
130                 }
131
132                 //RENDER STRINGS FOR OLED
133                 char line1[20];
134                 char line2[20];
135                 char line3[20];
136                 char line4[20];
137
138                 sprintf(line1, "Temp:␣%.1f␣C", temperature);
139                 sprintf(line2, "Humid:␣%.1f␣%%", humidity);
140                 sprintf(line3, "UV␣Ix:␣%.1f", uv_index);
141                 sprintf(line4, "O3␣V:␣%.2f␣V", actual_mq131_voltage);
142
143                 //PRINT TO SCREEN VIA SSD1306 LIBRARY
144                 ssd1306_Fill(Black);
145                 ssd1306_SetCursor(0, 0);
146                 ssd1306_WriteString(line1, Font_7x10, White);
147                 ssd1306_SetCursor(0, 16);
148                 ssd1306_WriteString(line2, Font_7x10, White);
149                 ssd1306_SetCursor(0, 32);
150                 ssd1306_WriteString(line3, Font_7x10, White);
151                 ssd1306_SetCursor(0, 48);
152                 ssd1306_WriteString(line3, Font_7x10, White);
153                 ssd1306_UpdateScreen();
154
155                 HAL_Delay(2000); // Refresh screen and poll every 2 seconds
156                 /* USER CODE END WHILE */
157
```

```
158                 /* USER CODE BEGIN 3 */
159         }
160         /* USER CODE END 3 */
161 }
162
163 /**
164  * @brief System Clock Configuration
165  * @retval None
166  */
167 void SystemClock_Config(void) {
168         RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
169         RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
170
171         /** Configure the main internal regulator output voltage
172          */
173         HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1_BOOST);
174
175         /** Initializes the RCC Oscillators according to the specified parameters
176          * in the RCC_OscInitTypeDef structure.
177          */
178         RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
179         RCC_OscInitStruct.HSIState = RCC_HSI_ON;
180         RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
181         RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
182         RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
183         RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV4;
184         RCC_OscInitStruct.PLL.PLLN = 85;
185         RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
186         RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
187         RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
188         if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
189                 Error_Handler();
190         }
191
192         /** Initializes the CPU, AHB and APB buses clocks
193          */
194         RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
195                         | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
196         RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
197         RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
198         RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
199         RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
200
201         if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK) {
202                 Error_Handler();
203         }
204 }
205
206 /* USER CODE BEGIN 4 */
207 uint32_t ADC_Read_Channel(uint32_t channel) {
208     ADC_ChannelConfTypeDef sConfig = {0};
209     sConfig.Channel = channel;
210     sConfig.Rank = ADC_REGULAR_RANK_1;                // Regular channel configuration for
        regular triggers
211     sConfig.SamplingTime = ADC_SAMPLETIME_47CYCLES_5; // Clear sampling window for reliable
        readings
212     sConfig.SingleDiff = ADC_SINGLE_ENDED;            // Configured for standard single-ended
         analog rails
213
214     // Apply the runtime register tracking configurations
215     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
216         return 0;
217     }
218
219     HAL_ADC_Start(&hadc1);
220     if (HAL_ADC_PollForConversion(&hadc1, 10) == HAL_OK) {
221         uint32_t value = HAL_ADC_GetValue(&hadc1);
222         HAL_ADC_Stop(&hadc1);
223         return value;
224     }
225
```

```
    HAL_ADC_Stop(&hadc1);
    return 0;
}
/* USER CODE END 4 */

/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void) {
        /* USER CODE BEGIN Error_Handler_Debug */
        /* User can add his own implementation to report the HAL error return state */
        __disable_irq();
        while (1) {
        }
        /* USER CODE END Error_Handler_Debug */
}
#ifdef USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

B.2.5. UV Sensor

This is the seperate code for reading the UV sensor. This section is also incorporated into the code in section B.2.4

```
    // READ UV SENSOR (Analog PA0 -> ADC1_CHANNEL_1 on STM32G4)
                uint32_t raw_uv = ADC_Read_Channel(ADC_CHANNEL_1);
                float uv_voltage = (raw_uv * 3.3f) / 4095.0f;
                float uv_index = uv_voltage / 0.1f;
```

B.2.6. Temperature and Humidity Sensor

This is the seperate code for reading the temperature and humidity sensor. This section is also incorporated into the code in section B.2.4

```
    // 1. Transmit register location to slave address
                if (HAL_I2C_Master_Transmit(&hi2c1, (0x40 << 1), &start_reg, 1, 100)
                                == HAL_OK) {
                        // 2. Receive the 4 sequential bytes back
                        if (HAL_I2C_Master_Receive(&hi2c1, (0x40 << 1), data, 4, 100)
                                        == HAL_OK) {
                                // Reconstruct 16-bit register words
                                uint16_t raw_temp = (data[1] << 8) | data[0];
                                uint16_t raw_humid = (data[3] << 8) | data[2];

                                // Apply Texas Instruments datasheet conversion curves
                                temperature = ((float) raw_temp / 65536.0f) * 165.0f - 40.0f;
                                humidity = ((float) raw_humid / 65536.0f) * 100.0f;
                        }
                }
```

C

Other Figures

C.1. LTSpice

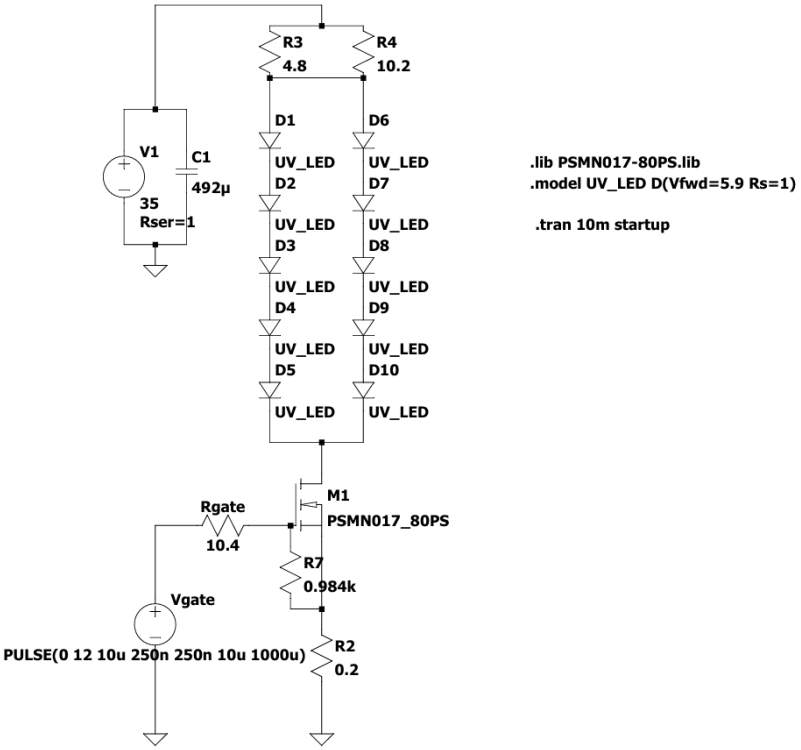


Figure C.1: PWM LTSpice Circuit

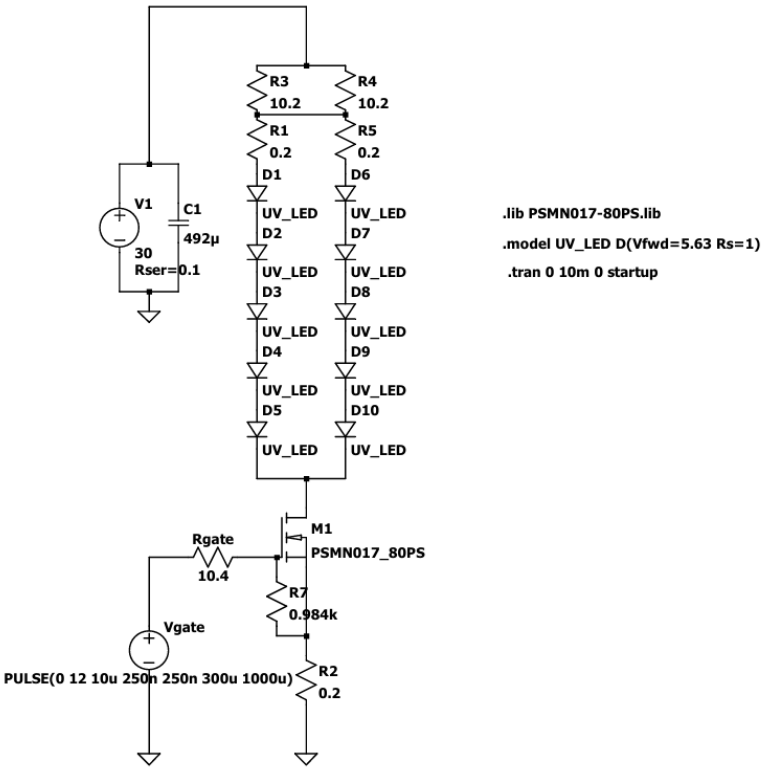


Figure C.2: PWM LTSpice Circuit Testing (without heatsink)

C.2. Schematics

C.2.1. Schematic of LED Panel

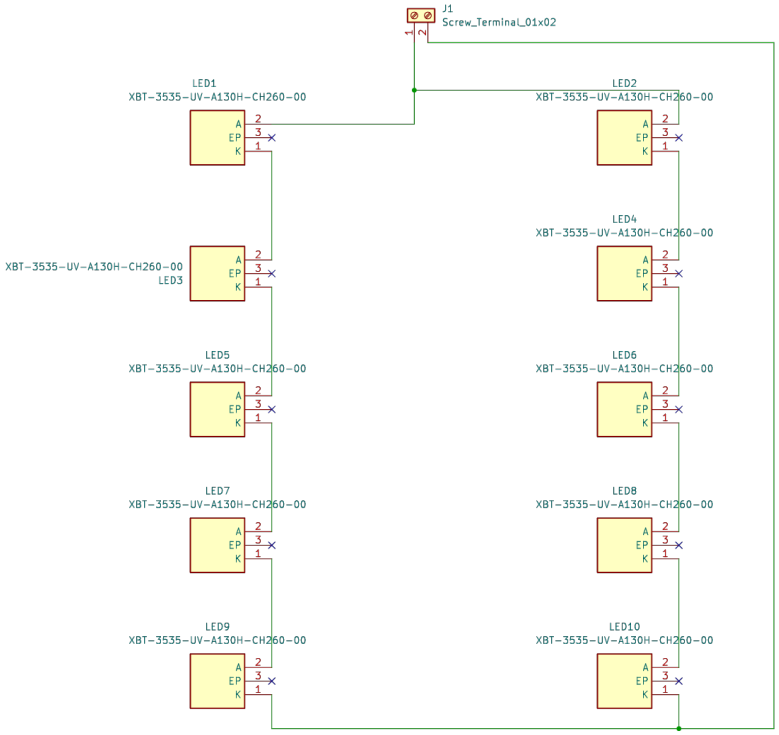


Figure C.3: Schematic of LED panel

C.3. PCB Designs

C.3.1. PCB Design of LED Panel

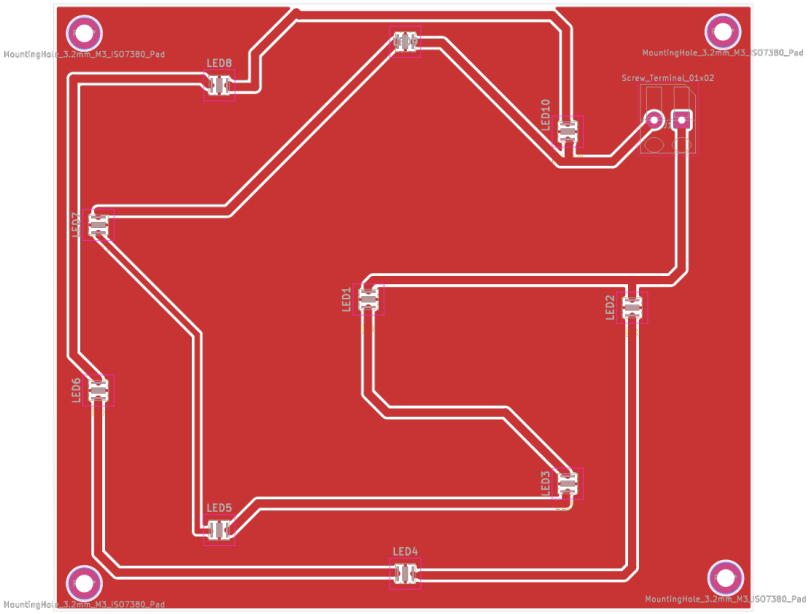


Figure C.4: Schematic of LED panel

C.4. Test set-up

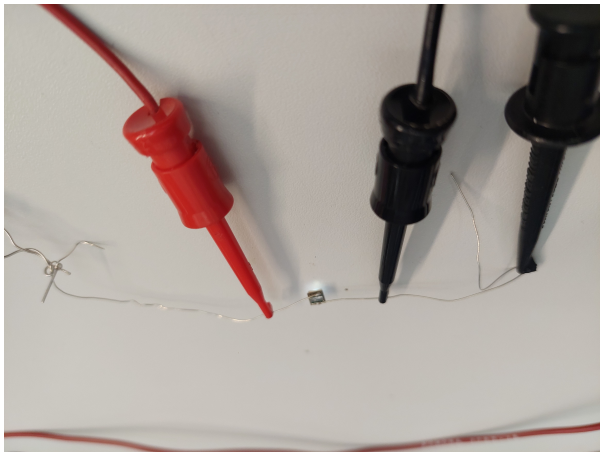


Figure C.5: Testing forward voltage over a singular LED

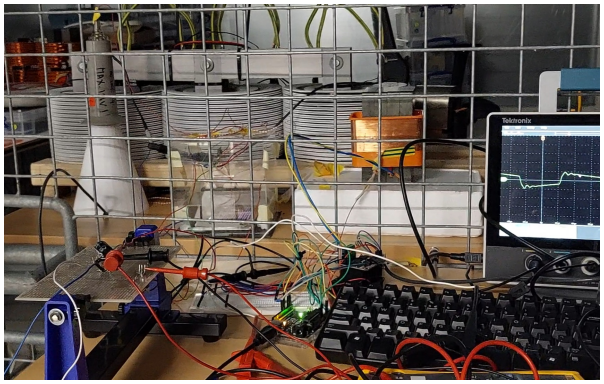


Figure C.6: Full broad integrated set-up

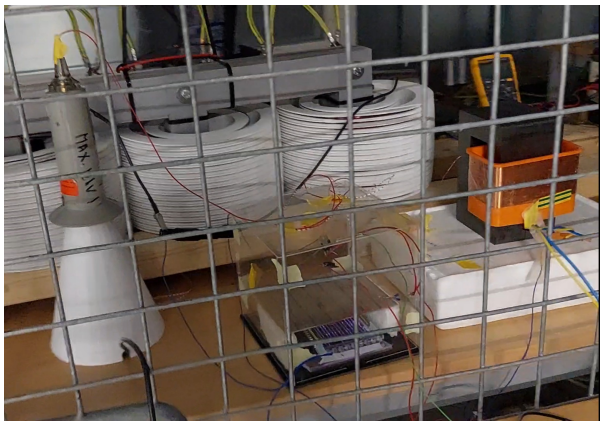


Figure C.7: Full integrated set-up

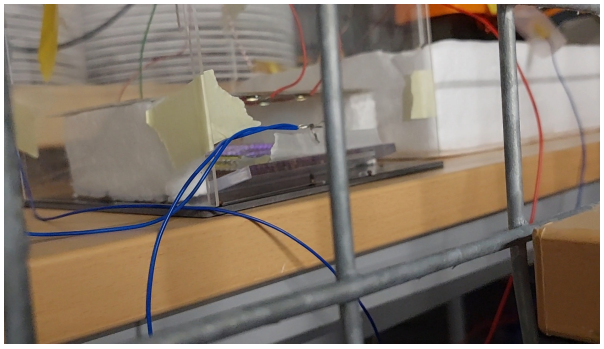


Figure C.8: Close-up box

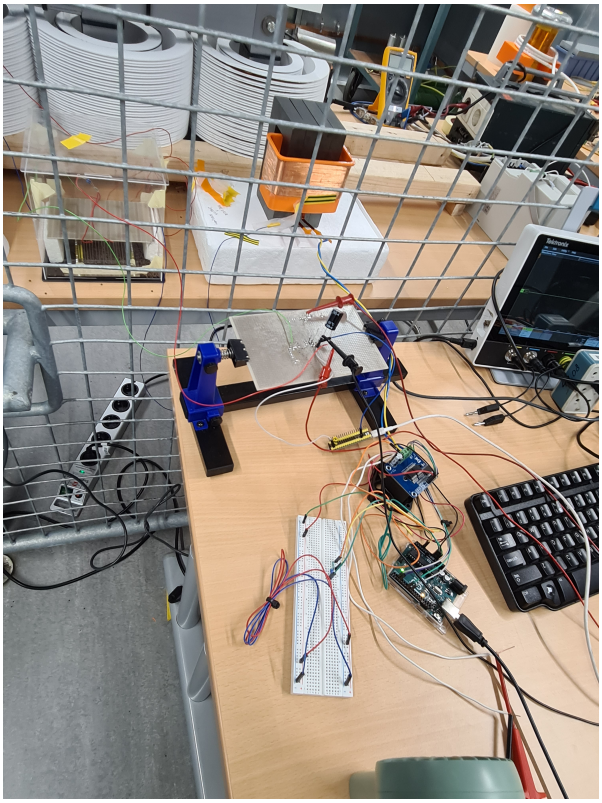


Figure C.9: Set-up general overview

C.5. Results

C.5.1. LTSpice Verification Measurements

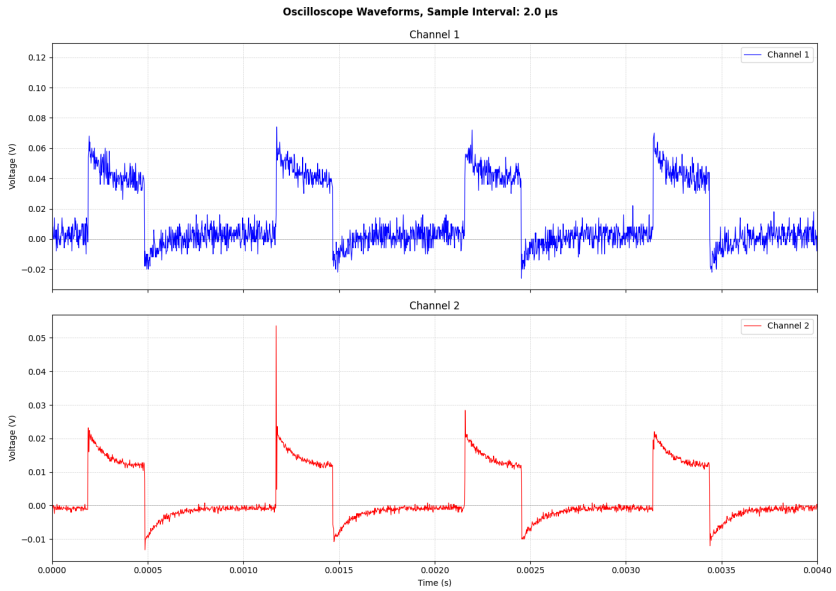


Figure C.10: Seperated LED Shunt Voltage Measurements

Note that for Figure C.10 although channel 1 seems more noisy than channel 2, that is at least partly because the two measurements had different volts-per-division settings, which caused the first channel to be more sensitive to noise.

C.5.2. Agar Tests

Intermediate Solo Agar Results

The results of the intermediate agar tests can be seen in Figure C.11. This test only assesses the effect of the treatment on Alternaria Brassicicola. In this figure, it can be seen that all duty cycles offer a substantial benefit when compared to the untreated sample. However none of the duty cycles have managed to fully eliminate the pathogen within 5 minutes with the frequency and current that was used during testing. With all duty cycles approximately 10% of the seeds have gotten infected.

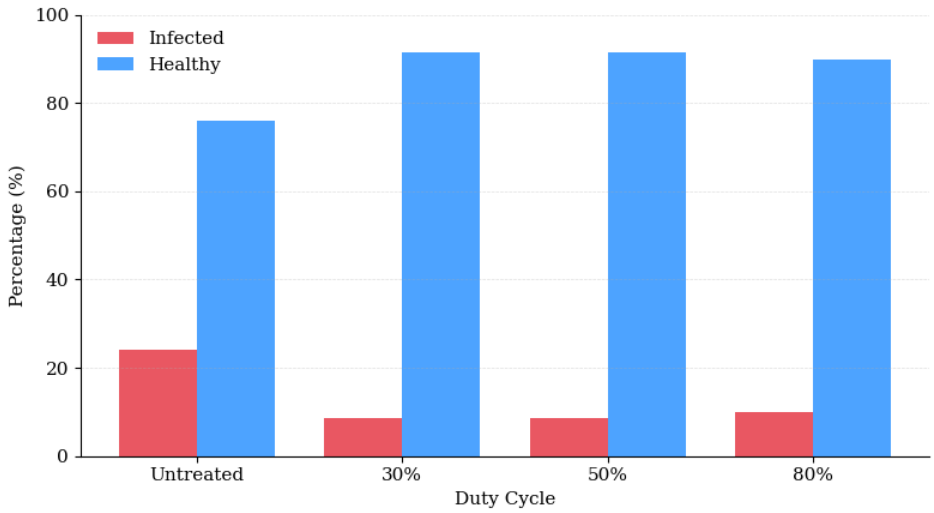
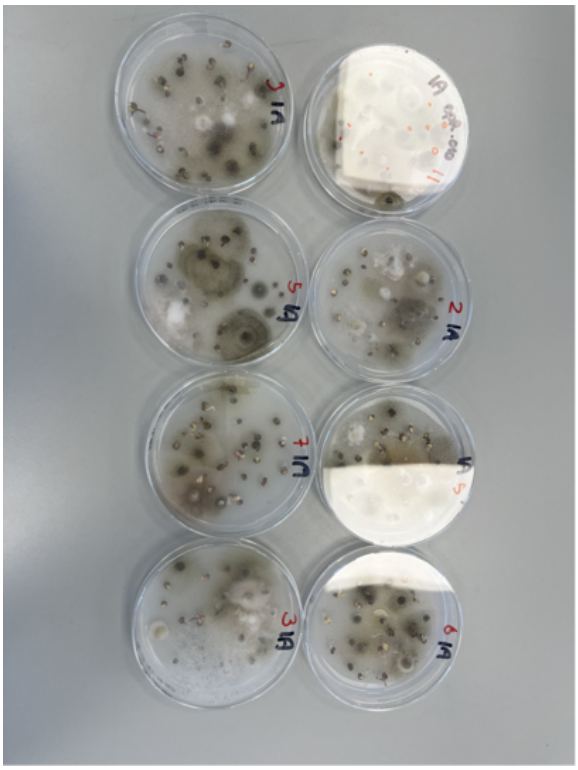
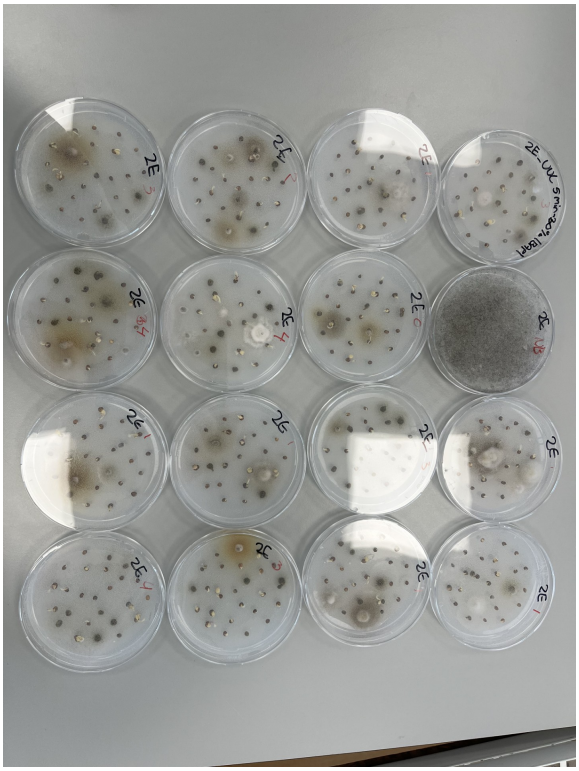


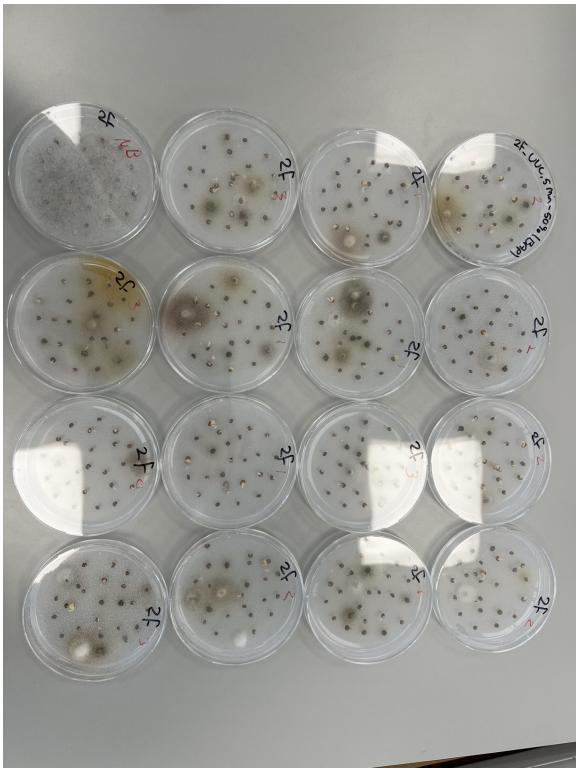
Figure C.11: Intermediate agar plate results for the bacterium Xanthomonas campestris pv. campestris after 5 minute UVC radiation with different duty cyles at frequency of 1 kHz and an average LED peak current of ≈ 165 mA



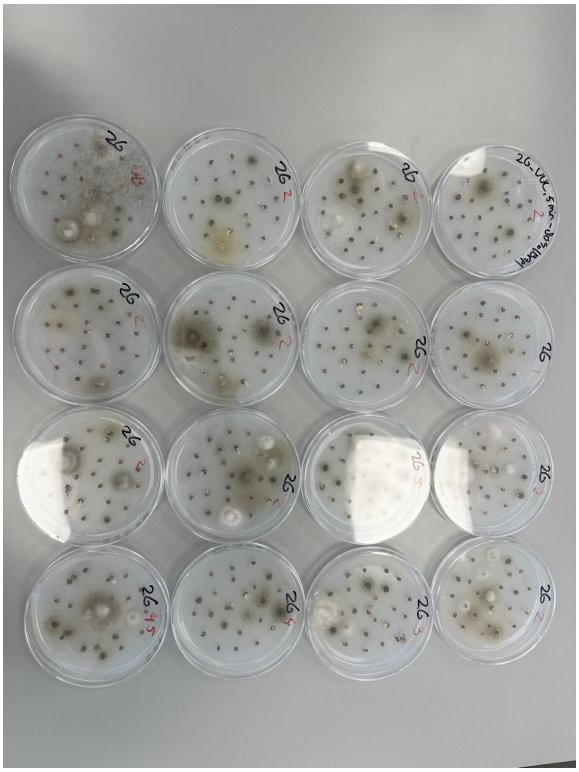
(a) Intermediate untreated agar plate



(b) Intermediate agar plate with seeds treated with 5 minute long 30% duty cycle, 1 kHz frequency, average 165 mA peak current UVC



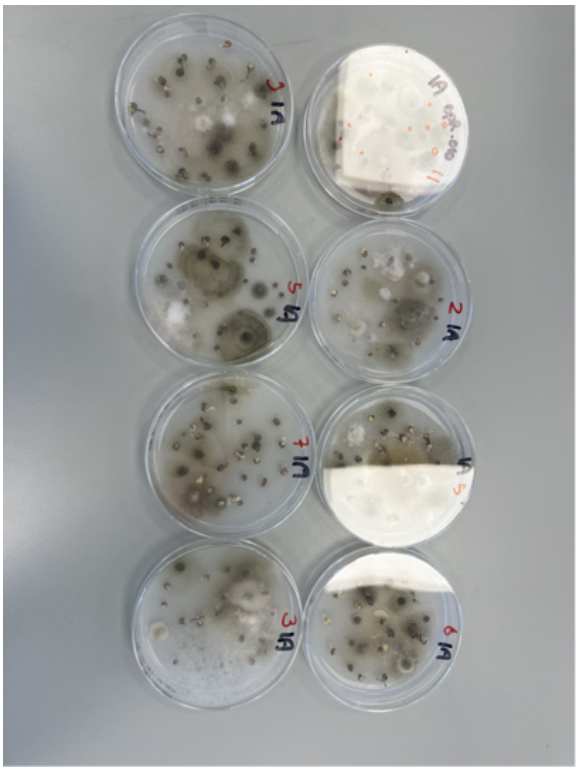
(c) Intermediate agar plate with seeds treated with 5 minute long 50% duty cycle, 1 kHz frequency, average 165 mA peak current UVC



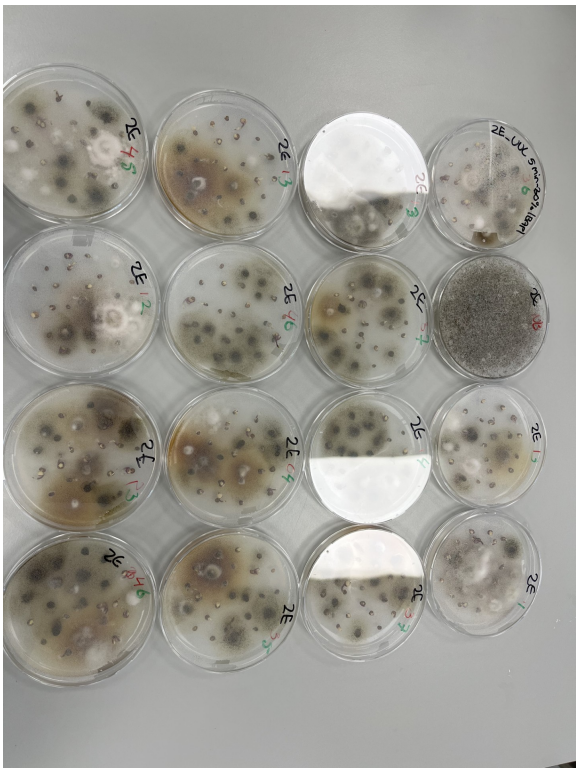
(d) Intermediate agar plate with seeds treated with 5 minute long 80% duty cycle, 1 kHz frequency, average 165 mA peak current UVC

Figure C.12: Intermediate agar plates

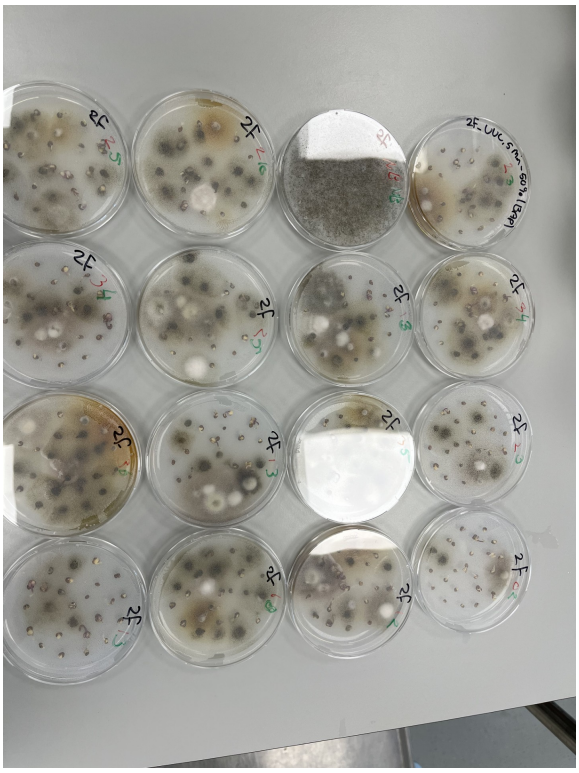
Final Solo Agar Results



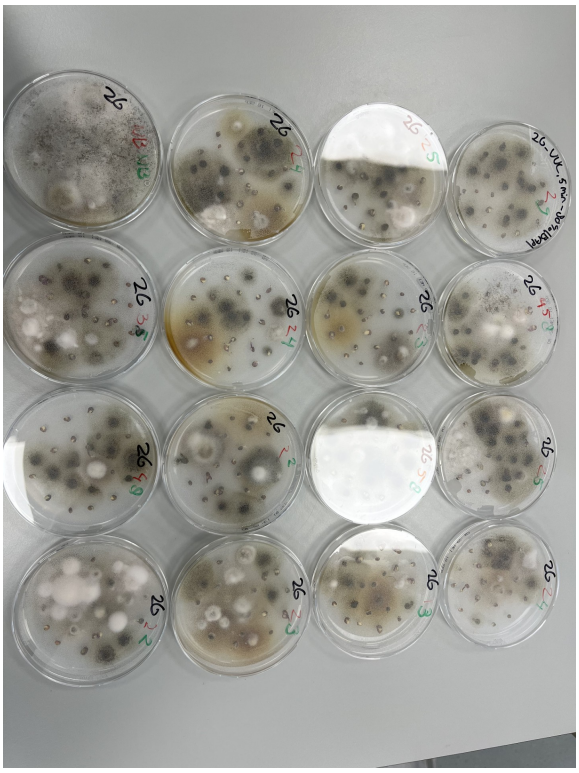
(a) Final untreated agar plate



(b) Final agar plate with seeds treated with 5 minute long 30% duty cycle, 1 kHz frequency, average 165 mA peak current UVC



(c) Final agar plate with seeds treated with 5 minute long 50% duty cycle, 1 kHz frequency, average 165 mA peak current UVC



(d) Final agar plate with seeds treated with 5 minute long 80% duty cycle, 1 kHz frequency, average 165 mA peak current UVC

Figure C.13: Final agar plates under different treatment conditions

C.5.3. Germination Test

The seeds were germinated by placing them on top of paper towels and kept humid by spraying them with water daily.



Figure C.14: Seeds after 5 days

D

Tables

D.1. PCR Results

D.1.1. UV

Xcc

Treatment	Cq (No PMA)	Cq (With PMA)
UVC 30% Duty Cycle (Rep 1)	27.24	32.58
UVC 30% Duty Cycle (Rep 2)	28.9	35.94
UVC 50% Duty Cycle (Rep 1)	28.38	32.35
UVC 50% Duty Cycle (Rep 2)	30.81	35.86
UVC 80% Duty Cycle (Rep 1)	29.86	34.91
UVC 80% Duty Cycle (Rep 2)	25.02	29.03
Control (Rep 1)	29.57	32.8
Control (Rep 2)	28.21	30.19

Table D.2: qPCR results for Xcc under different UVC duty cycle treatments (1 kHz, 5 min exposure, 165 mA peak current). Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

Abco

Treatment	Cq (No PMA)	Cq (With PMA)
UVC 30% Duty Cycle (Rep 1)	22.83	29.75
UVC 30% Duty Cycle (Rep 2)	23.28	30.03
UVC 50% Duty Cycle (Rep 1)	23.2	28.84
UVC 50% Duty Cycle (Rep 2)	24.31	30.74
UVC 80% Duty Cycle (Rep 1)	23.9	30.64
UVC 80% Duty Cycle (Rep 2)	23.65	30.08
Control (Rep 1)	22.45	27.97
Control (Rep 2)	23.08	29.39

Table D.4: qPCR results for Abco under different UVC duty cycle treatments (1 kHz, 5 min exposure, 165 mA peak current). Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

D.1.2. UV and Plasma

Xcc

Treatment	Cq (No PMA)	Cq (With PMA)
UVC and plasma 5 min (Rep 1)	31.74	37.42
UVC and plasma 5 min (Rep 2)	28.11	36.66
UVC and plasma 10 min (Rep 1)	28.84	35.53
UVC and plasma 10 min (Rep 2)	28.11	41.46
UVC and plasma 20 min (Rep 1)	29.59	38
UVC and plasma 20 min (Rep 2)	29.5	39.43
Control (Rep 1)	31.5	35.32
Control (Rep 2)	29.23	32.78

Table D.6: qPCR results for Xcc under different time samples using UVC and plasma. Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

Abco

Treatment	Cq (No PMA)	Cq (With PMA)
UVC and plasma 5 min (Rep 1)	24.9	30.55
UVC and plasma 5 min (Rep 2)	22.71	29.75
UVC and plasma 10 min (Rep 1)	22.51	27.8
UVC and plasma 10 min (Rep 2)	22.14	28.85
UVC and plasma 20 min (Rep 1)	25.49	32.46
UVC and plasma 20 min (Rep 2)	22.74	31.86
Control (Rep 1)	22.05	28.89
Control (Rep 2)	23.44	29.87

Table D.8: qPCR results for Abco under different time samples using UVC and plasma. Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

Xcc with domancy

Treatment	Cq (No PMA)	Cq (With PMA)
UVC and plasma 5 min (Rep 1)	30.47	38
UVC and plasma 5 min (Rep 2)	29.3	36.04
UVC and plasma 10 min (Rep 1)	30.57	38.55
UVC and plasma 10 min (Rep 2)	29.8	38
UVC and plasma 20 min (Rep 1)	26.58	33.49
UVC and plasma 20 min (Rep 2)	30.03	36.88
Control (Rep 1)	28.39	37.88
Control (Rep 2)	31.04	36.95

Table D.10: qPCR results for Xcc under different time samples using UVC and plasma after seeds were sprayed with water for dormancy. Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

Abco with dormancy

Treatment	Cq (No PMA)	Cq (With PMA)
UVC and plasma 5 min (Rep 1)	24.25	31.11
UVC and plasma 5 min (Rep 2)	23.68	29.35
UVC and plasma 10 min (Rep 1)	23.69	30.1
UVC and plasma 10 min (Rep 2)	23.52	29.66
UVC and plasma 20 min (Rep 1)	23.57	30.18
UVC and plasma 20 min (Rep 2)	24.58	31.81
Control (Rep 1)	22.15	28.11
Control (Rep 2)	24.03	31.24

Table D.12: qPCR results for Abco under different time samples using UVC and plasma after seeds were sprayed with water for dormancy. Two repetitions per treatment sample. Fluorescence was measured using FAM dye.

D.2. Agar Results

D.2.1. UV

D.2.2. Intermediate

Treatment	Infected	Healthy	Total Seeds	Percentage Infected	Percentage Healthy
UVC 30% Duty Cycle	48	152	200	24.0	76.0
UVC 50% Duty Cycle	32	343	375	8.5	91.5
UVC 80% Duty Cycle	32	343	375	8.5	91.5
Control	38	337	375	10.1	89.9

Table D.14: Intermediate (6 days) agar results for Abco under different UVC duty cycle treatments (1 kHz, 5 min exposure, 165 mA peak current).

D.2.3. Final

Treatment	Infected	Healthy	Total Seeds	Percentage Infected	Percentage Healthy
UVC 30% Duty Cycle	65	310	375	17.3	82.7
UVC 50% Duty Cycle	66	309	375	17.6	82.4
UVC 80% Duty Cycle	73	302	375	19.5	80.5
Control	48	152	200	24.0	76.0

Table D.16: Final (9 days) agar results for Abco under different UVC duty cycle treatments (1 kHz, 5 min exposure, 165 mA peak current).

D.2.4. UV and Plasma

Results to be added after submission due to results not being received yet.

E

Calculations

E.1. PWM Circuit

E.1.1. LEDs

Dynamic Resistance of LEDs

Using the forward voltage shift plot from the datasheet of the LEDs which can be seen in Figure E.1, a crude estimation was made for the dynamic resistance of the LEDs by taking the slope of the curve at 800 mA.

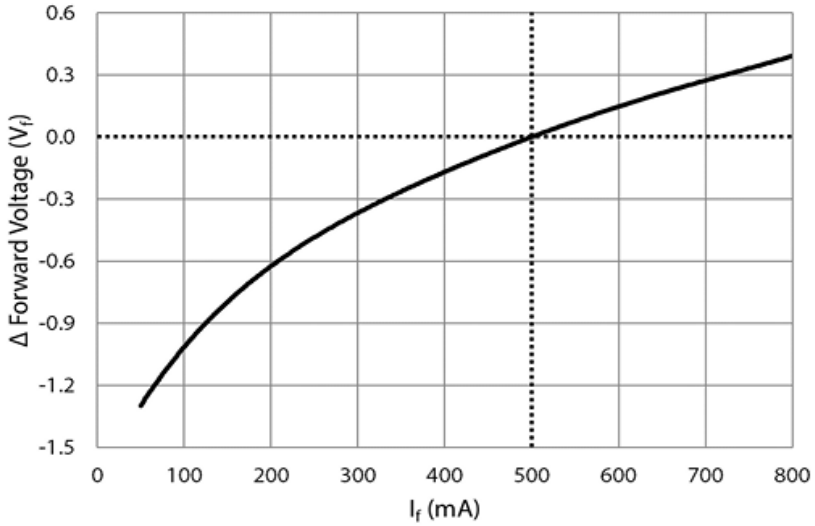


Figure E.1: Datasheet Plot for Forward Voltage Shift. Reproduced from [59].

$$R_d = \frac{\Delta V}{\Delta I} \approx \frac{0.3}{0.3} = 1.0\Omega$$

E.1.2. Average LED Forward Voltage

The average LED forward voltage (V_f) was calculated by taking the average current drawn by the power supply towards the LEDs and calculating the peak current, assuming the current that goes into each LED string is divided equally. The V_f was iteratively adjusted until that average peak current was achieved in the LTSpice simulation.

Using the following system parameters: 1 kHz frequency, 30% duty cycle. The average current drawn by the power supply was approximately 0.1 A. With a duty cycle of 30% and the assumption the current

would be equally drawn by both LED string, that translates to an approximate peak current draw of $\frac{0.1}{0.3\times 2} \approx 167$ mA. This approximate peak current was achieved when the V_f of the LED model was iteratively adjusted to 5.63 V.