Design of Hardware Imaging System for ddLAMP-based Microbial Source Tracking (Dec 2024)

Sana Hasan, *Wayne State University*

***Abstract*— This thesis presents the design, implementation, and testing of a hardware imaging system for droplet digital Loop-Mediated Isothermal Amplification (ddLAMP) used in microbial source tracking in wastewater. The system integrates a Raspberry Pi Zero 2 W microcomputer with various peripherals to automate and control heating, illumination, and imaging of fluorescence-based reactions. The ddLAMP procedure requires maintaining the sample at a steady temperature of 60 °C, achieved using a thermoresistive foil heater regulated by a PID or duty-cycle-based controller, and monitored by a thermocouple amplifier circuit (MAX31856). For excitation of the fluorescent dye in the sample, a custom-designed ring-light Printed Circuit Board (PCB) embedded with SPI-controlled DotStar (SK9822) LEDs emits light in the 450–470 nm wavelength range. A high-sensitivity Sony IMX477 sensor of 12 megapixels captures images of droplets on a cell-well chip, with exposure parameters tuned for low-light conditions. Custom Python scripts developed by the author manage temperature control, LED lighting, and image capture. Integration with a mobile app enables wireless initiation of tests, including automated image transfer and geolocation tagging. The system's design allows for reliable, reproducible imaging while being low-cost and portable. The prototype discussed demonstrates functionality across hardware and software components, with results showing promising droplet distinction under fluorescence. Further controlled lab testing is recommended, to optimize heating stability under real-world conditions and to test a finalized fluorescence formula.**

***Index Terms*— Microcomputers, Bluetooth, Pulse Width Modulation, Serial Peripheral Interface, Printed Circuit Boards, Programming, Hardware, Imaging, Image Processing, Microfluidics, Heating Systems, Prototypes, Wastewater.**

I. Introduction

This thesis presents the design process and development of a portable ddLAMP imaging system developed for fluorescence-based microbial source tracking in wastewater. The system is presented in a chronological breakdown. Following this high-level description are the details of each sub-system the author worked on with a step-by-step analysis. Some components were developed collaboratively; their integration points with my design process are described in the overview.

II. Overall system design

The purpose of this system is to prepare and image a sample of wastewater in order to determine the amount of waste in the water. This system was developed through an interdisciplinary collaboration between Electrical Engineering and Chemistry at Wayne State University. The project was conducted under the supervision of Professor Basu, with imaging and analysis methods based on his algorithms, and in coordination with the chemistry team led by Professor Ram. The sample must first be mixed with a chemical compound that will react with DNA and become fluorescent upon this reaction. Then it is poured into a cell-well chip that evenly distributes tiny portions of the liquid.

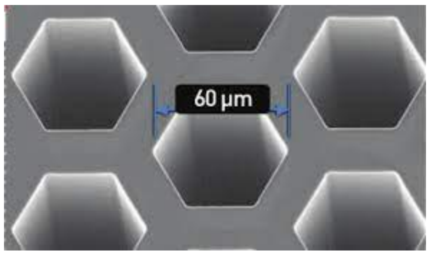


Fig. 1: The QuantStudio® 3D digital PCR cell-well Chip [1]

This chip is then placed onto an aluminum block within the 3d-printed housing of the system. This block houses a heating element and thermocouple to get the sample to 60 degrees Celsius and maintain this temperature throughout the trial. Then, LEDs are set to the right color in the 450-470 nm range, and adequate brightness to 'excite' the fluorescence of the solution. The housing closes around the sample chip to provide a dark environment so the lighting is fully controlled by these LEDs. The image sensor connected to the Pi then takes a picture with calibrated exposure settings to optimize droplet distinction and clarity.

The divisions between each cell well and different levels of fluorescence must be clearly defined for processing by computer vision algorithms. For this use case, these algorithms analyze the number of fluorescent cells and brightness to measure the level of waste in the water [2].

Manuscript received December 18, 2024. This work was funded and supported by the Great Lakes Water Authority. This work was completed in the laboratory of Professor Amar Basu, at Wayne State University.

Sana Hasan is with the Electrical Engineering Department, Wayne State University Detroit, MI 48202 USA. (e-mail: sanabuu@wayne.edu).

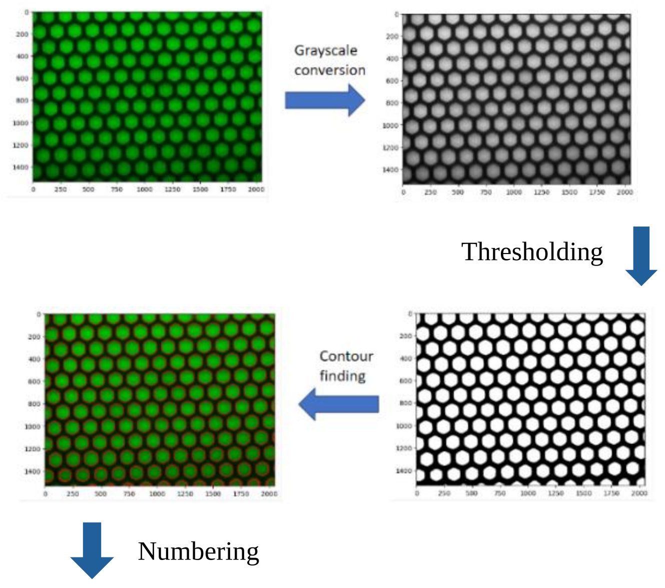
The Raspberry Pi microcomputer runs code written in Python for three main functions; the temperature control loop, the excitation LEDs, and the imaging.

The housing contains the temperature control system, image sensor, color filter, LED lighting ring, and the Raspberry Pi. This microcomputer is Bluetooth and WiFi enabled allowing it to communicate with an external device.

A Capstone project of a team of four computer-science students developed a mobile app, with me as the primary lab correspondent. The app's purpose is a visually appealing and simple user interface to initiate the tests, adjust variables, and store results on a database as well as on the device. Professor Basu provided initial design guidelines, which we discussed as a group. The Capstone team developed the mobile app based on those discussions, and the author tested its functionality in the lab.

I developed Python scripts to run specific functions on the Pi, as well as designing separate hardware/circuitry that would connect physically to the microcomputer. The app was programmed to run on a phone and connect to the Pi using Bluetooth, run Python programs from pressing buttons on the GUI (graphical user interface), and connect to a database to deposit images taken from the Pi.

Image processing algorithms will then be applied to the images taken, which would also be in Python using OpenCV libraries. As the following figures show, it would detect the fluorescent cells, rate the fluorescence, and number them to output a percentage that represents the level of waste in the water sample [2].



(figure continued on next column)

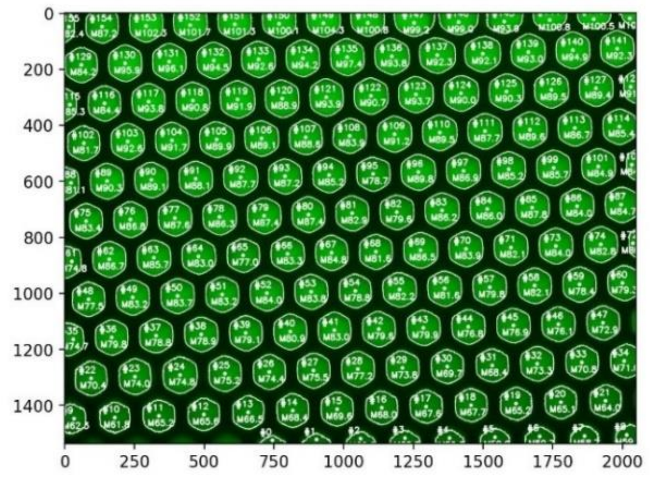
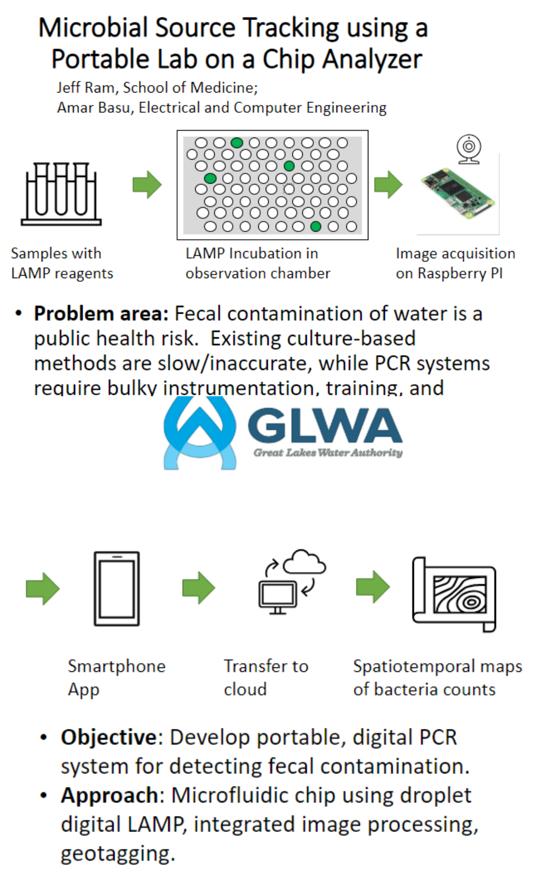


Fig. 2: Cell-Well Chip Image Processing

A. Problem and Objectives Diagram:



Microbial Source Tracking using a Portable Lab on a Chip Analyzer

Jeff Ram, School of Medicine;
Amar Basu, Electrical and Computer Engineering

Samples with LAMP reagents
LAMP Incubation in observation chamber
Image acquisition on Raspberry PI
GLWA
Great Lakes Water Authority
Smartphone App
Transfer to cloud
Spatiotemporal maps of bacteria counts

- **Problem area:** Fecal contamination of water is a public health risk. Existing culture-based methods are slow/inaccurate, while PCR systems require bulky instrumentation, training, and

- **Objective**: Develop portable, digital PCR system for detecting fecal contamination.
- **Approach**: Microfluidic chip using droplet digital LAMP, integrated image processing, geotagging.

Fig. 3: Problem, Objectives, Approach diagram

B. Table I: Table of system specifications:

Subsystem	Description	Specifications
Housing	Overall	contains each subsystem
		small enough to transport by carrying
		easy to load with chip
		Heat-safe material to 65 °
		Provides thermal insulation
		Blocks external light for a low-light imaging environment
	3D-printed material	Designed Using AutoCad
		Phrozen Aqua 8k 3D Printing Resin
Microcont-roller/comp-uter	Overall	Able to control the necessary modules with code
		Adequate processing power to run image analysis algorithms
		Small form factor and cheap
		Ability to communicate with Wi-Fi and/or Bluetooth
	Raspberry Pi Zero 2 W	64-bit ARM Cortex-A53 Processor
		512 MB of SDRAM
		64 GB microSD card containing Debian Linux Operating System
		2.4GHz 802.11 b/g/n wireless LAN (WiFi)
		Bluetooth 4.2 + BLE
		Output to mini-HDMI
		Powered by 5V microUSB
		Additional microUSB I/O slot
		Dedicated CSI-2 slot for camera cable input
		40 Input/Output (IO) Pins
		IO includes General-Purpose (GPIO), 5V, 3.3V, GND, and several protocols:
		Pulse-Width Modulation (PWM),
		Serial Peripheral Interface (SPI),
		Inter-Integrated Circuit (I2C),
		Universal Asynchronous Receiver / Transmitter (UART)

Subsystem	Description	Specifications
Imager Module	Sony IMX477R sensor	6.3x4.7mm back-illuminated CMOS sensor
		12.3 megapixel resolution
		size
		C/CS mount type (similar to microscope)
		Low Light performance
	Kodak Wratten #12 gelatin filter	Low profile
		Blocks out extraneous blue light to properly see fluorescense
	100x Microscope Lens	Magnification 0.12X ~ 1.8X
		Zoom Ratio 15:01
		3 mm minimum object distance
	Aperture	not changeable through software
	ISO	800 (max sensitivity for low light imaging)
	Shutter	Images 1 - 5 s
		video preview 200 ms - 1 s
Light-Emitting Diode (LED) controller	Overall	Have a light Spectra between 450 - 470 nm to excite fluorescense in wastewater solution
		Small form factor
	Adafruit DotStar LEDs	color (8 bits per Red, Green, and Blue)
		Adjustable 5-bit brightness (32 Levels)
		5mm x 5mm form factor
		quick 2-wire SPI for communication
	Printed Circuit Board (PCB)	Provides the proper power and connection to the LEDs
		Ground Plane
		Thicker traces for proper power rail
		Ring-Light form factor for even illumination
		Circular with hole in middle for camera lens
		Header pin-friendly through-holes for easy wiring to the Pi
		70 mm diameter
Power Supply	Battery Life	well-reviewed long life of several hours
	Output	12 max 9 min Voltage
		Amps

Subsystem	Description	Specifications
Temperature Controller	Overall	Provide 60 ° Celsius ± 3 ° to the chip over 60 - 90 min
	Thermoresistive Foil Heater	good heat transfer performance
		Thin and flexible
		25 mm x 50 mm
	2N3906 PNP BJT	Junction Silicon Bipolar Transistor
		part of level shifter
	IRLB8721 MOSFET	MOS Field-Effect Transistor
		Low Threshold Voltage (VGS-th) of 1.35V-2.35V
		High Drain-Source Voltage (VDS) of 30V
		Commonly used with microcontrollers
	Resistors	220 Ω Resistor
		1.8 kΩ Resistor
		~1kΩ pull-down Resistor
	Thermocouple	K-Type Glass Braid Insulated
	MAX31856 Thermocouple Amplifier	Adafruit Universal Thermocouple Breakout
		Compatible with 3.3V
		Built-in Analog-Digital Converter (ADC) for digital temperature data
		4-wire SPI communication
		resolution of range -20°C to +85°C
		Built-in Fault Detection and Over-Input Protection
		Error of ± 0.15%
		Average of 100 ms temperature conversion time

Specifications of Raspberry Pi and the Raspberry Pi High Quality Camera (image sensor) are from the official Raspberry Pi Ltd. Site; [3] and [4]. The Dotstar LED information is from the datasheet [5]. The battery pack was a lab resource, and the information was read off the label. Kodak filter information is found at [6]. Lens was likely ordered from Amazon [7]. Heater is polyimide based, found [8]. 2N3906 and IRLB8721 Datasheets [9] and [10]. Thermocouple amplifier datasheet [11] found at [12] which links to the thermocouple as well.

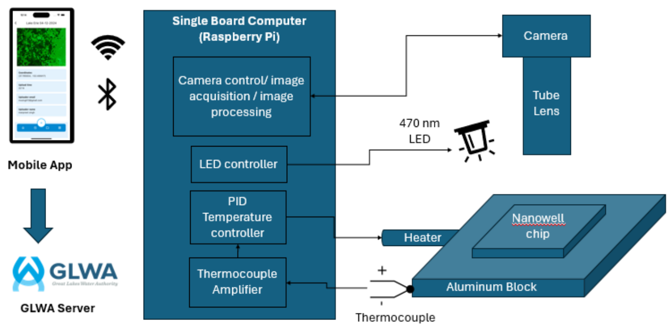


Fig. 4: ddLAMP System Block Diagram

This diagram shows how the Raspberry Pi, Camera, temperature controller, LED controller, Bluetooth interface and interact with each other.

C. *System Assembly and Housing*

The prototyping faced mechanical challenges and funding limitations for the 3D printing. The in-lab 3d printer broke and was under maintenance for a while, and the other accessible 3d printers across campus took more time and cost. Because of this, the following images are of a prototype housing with some assembled pieces and not the final piece.

Here is the computer drafted design for the more finalized housing that was not printed as of the time of this paper.

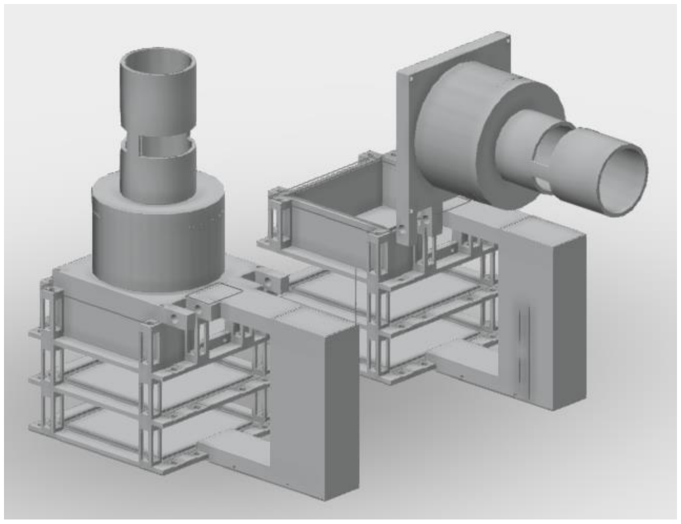


Fig. 5: AutoCAD Drafted Design for the Housing

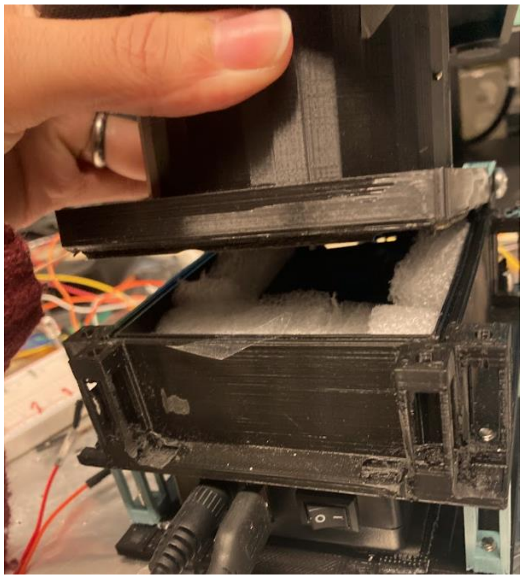


Fig. 6: Image of Housing

Here is a cross-section above showing insulation surrounding the metal block with the chip placed on it. This image also shows the open section below that has the batter pack which provides 5V to run the Pi from the USB slot, and to the heater circuit from the other 12V output. The image below shows the fully open prototype with the attached pcb for the LED. In the final system, the hole with the wires leading out would go into a handle containing the Pi, Thermocouple amplifier board, and heater control board. This would allow the seamless low-light environment.

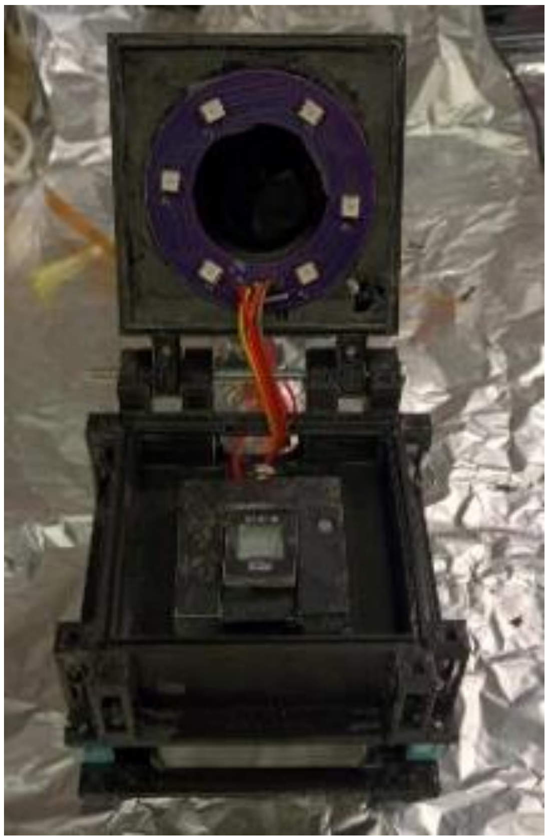


Fig. 7: Image of Housing showing LED ring light

This image below shows the full housing with my hand for scale. This image also shows the camera sensor and lens as it feeds into the housing. The handle in the CAD drawing shows how it would not change the form factor from this printed prototype too much and provide easy carrying. The one piece not shown would be a screw-on cover to hide and secure the camera sensor and lens, and this would connect down to the handle to protect the camera cable.

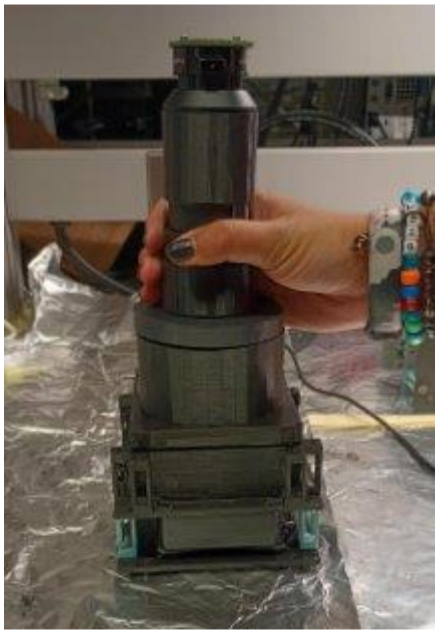


Fig. 8: Image of full Housing

III. Raspberry Pi module Implementation

A. The Choosing of the Raspberry Pi Zero 2 W

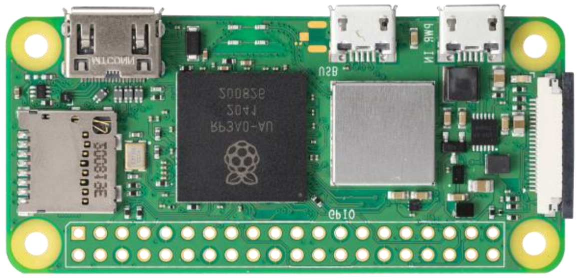


Fig. 9: Image of the Raspberry Pi Zero 2 W [3]

In the system specifications table, many details are summarized about the controlling microcomputer for reference. The overall specifications necessary for the project are described, and relevant product specifications of the selected board are outlined.

An alternative considered was the Arduino microcontroller [13]. Arduino has a dedicated Integrated Development Environment (IDE) that runs on a separate computer to develop and upload code to the Arduino. Indeed, many of our components were sourced from Adafruit, a manufacturer that has the highest compatibility with Arduino boards. The company provides libraries easy to add in this IDE, and so these boards are commonly used in these types of projects.

The classification of Arduino as a microcontroller is due to programming on a separate computer, after which the microcontroller may independently run when powered.

The Raspberry Pi uses an operating system (OS) providing a Linux environment, categorizing this board as a microcomputer. This enables control of all of the aspects of the system and is configurable and programmable as a standalone device. The Raspberry Pi chosen, the Zero 2 W, has built-in Wi-Fi and Bluetooth components; Arduino boards require additional components to implement communication over these protocols. Adafruit libraries for the Raspberry Pi are still available, and must be installed to the pi and then called from the code

B. *Detailed diagrams of Raspberry Pi*

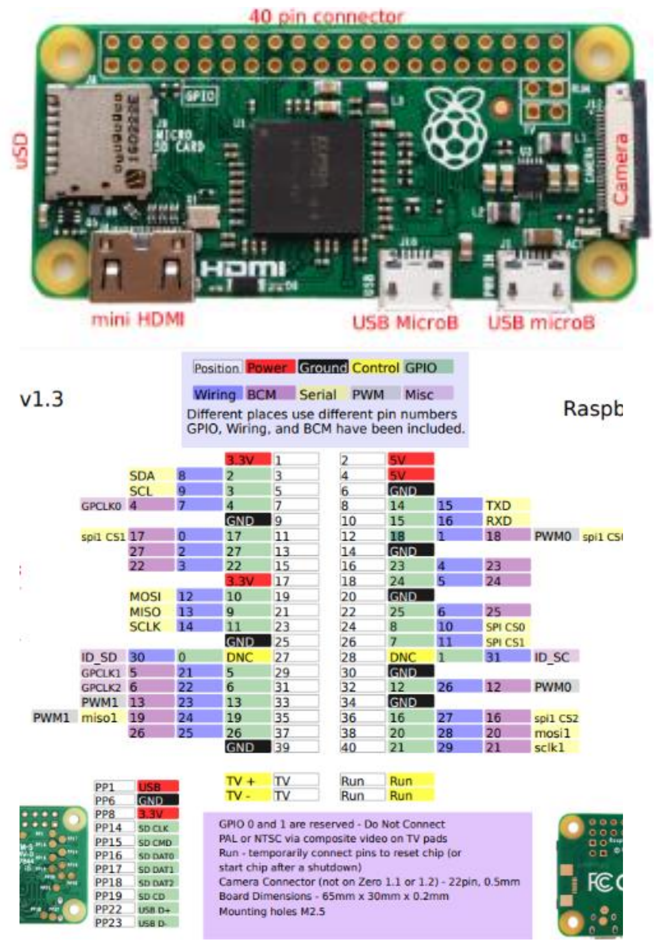

40 pin connector
uSD
Camera
mini HDMI
USB MicroB
USB microB
v1.3
Raspb
Position Power Ground Control GPIO
Wiring BCM Serial PWM Misc
Different places use different pin numbers GPIO, Wiring, and BCM have been included.
GPIO 0 and 1 are reserved - Do Not Connect
PAL or NTSC via composite video on TV pads
Run - temporarily connect pins to reset chip (or start chip after a shutdown)
Camera Connector (not on Zero 1.1 or 1.2) - 22pin, 0.5mm
Board Dimensions - 65mm x 30mm x 0.2mm
Mounting holes M2.5

Fig. 10: Detailed I/O Diagram [14]

The first diagram is split into a couple of the most relevant portions of it for better viewability [14]. This shows the different functionalities of the GPIO ports other than the GPIO (camera, USB). Among the color-coded numbers, BCM and Wiring are referencing specific libraries (Python-based, possibly also have MATLAB versions). In most libraries, the green GPIO number is what is used to address the header to a certain function.

Serial refers to SPI, which uses 2- to 4-wire communication and there are two sets of these on the Pi [15]. We use both in our system. Through this communication protocol, some peripherals can connect to the same set of SPI if the CS (Chip Select) is used. This pin tells the device to 'wake up' to transfer data, and each device under SPI would have a different CS pin and use the same other wires for the data. The usage of the Clock is necessary, the MISO (main in, sub out) is for when data is coming into the Pi from a device, and the MOSI (main out, sub in) is for when data is transmitted.

Two-wire SPI is used for the LEDs, the clock and data out lines. As such, that instance of the SPI must be dedicated to the LEDs, but the data in (MOSI) can be reallocated as it is still usable under the GPIO [5]. The only other thing we use the SPI for is the thermocouple amplifier, which uses all four wires [11]. Both of these systems also do require a power supply and a ground as well.

PWM simply refers to the ability of that pin to transmit using Pulse-Width Modulation. There is specific PWM hardware allowing the signals with the rise/fall to be precisely timed, but less precise PWM is possible with any general-purpose I/O pin. PWM is used to control the heater more precisely. A parameter referred to as the duty cycle represents how much of a certain length of time a square wave is 'on' versus how much it is 'off'.

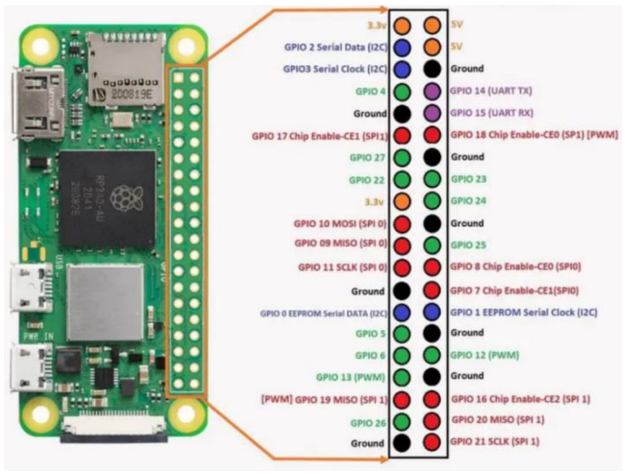


Fig. 11: GPIO pins Diagram [16]

This diagram reiterates information on GPIO mapping. The 40 GPIO headers are displayed in an easier to read diagram [16]. There is an error here, where GPIO 20 / MISO is mislabeled and should be MOSI. Thus, cross-referencing is required between both diagrams throughout the design process.

C. *Wiring and Connection Hardware Overview*

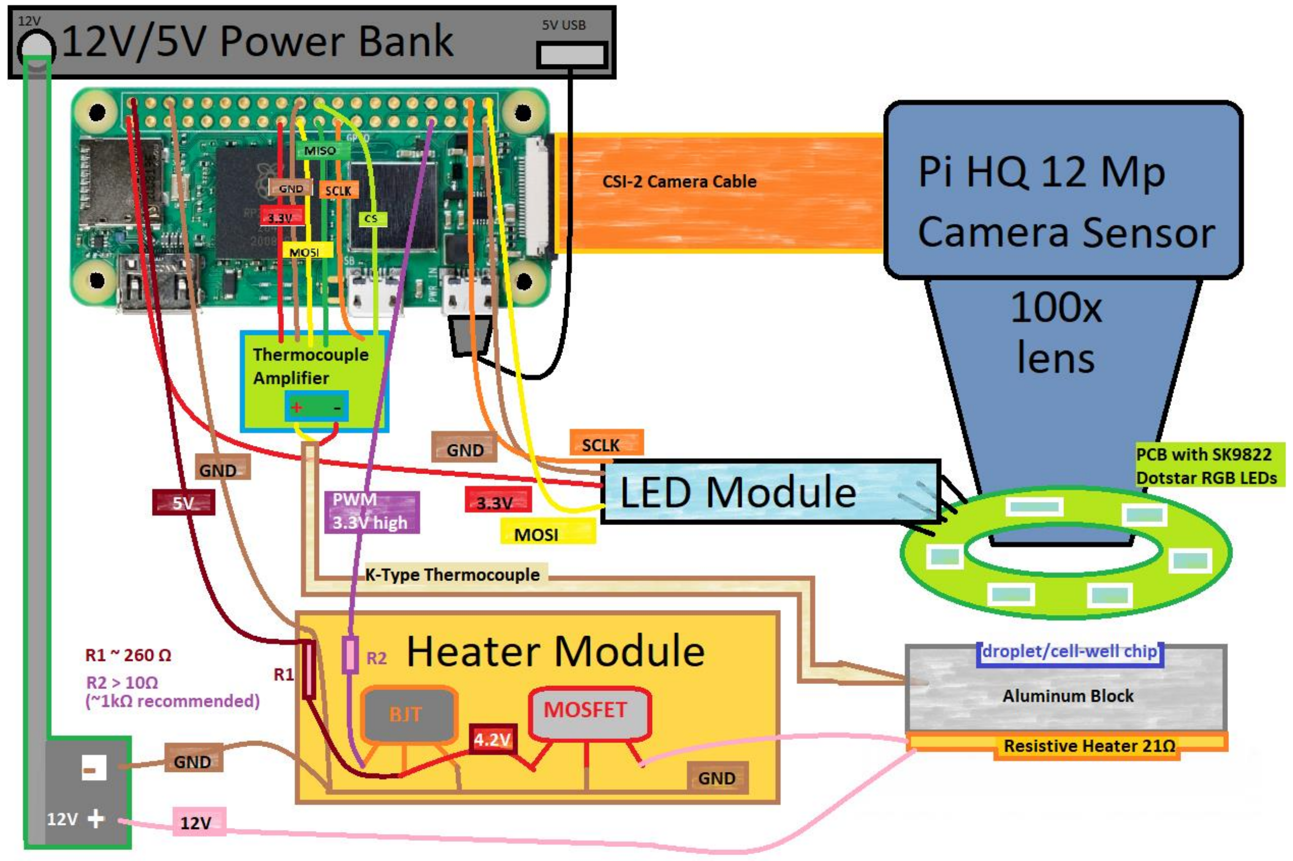


Fig. 12: Detailed Hardware Overview Diagram

This Diagram shows the connections on the hardware. Connections include: 1) a heater module which includes: a) a 12V supply; b) a switch circuit; and c) a resistive heater; 2) a temperature measurement module consisting of a thermocouple and amplifier; 3) an LED controller module, and 4) a camera module. The heater and temperature measurement are both part of the Temperature Controller.

D. *Software Overview*

On the Raspberry Pi, the programming language that has the most support is Python. We have mentioned Adafruit's libraries in the Arduino IDE; for general use with microcontrollers and RPIs, Adafruit also developed the Python framework of CircuitPython [17].

This collection of libraries gives hardware support for myriad components and applications. The Blinka library in particular is used to provide low-level hardware APIs [17]. API stands for Application Programming Interface, something that provides an interface to allow applications programmed with the API to easily access features of an OS or another service.

Blinka converts the higher-level Python code to whatever library the hardware provides- including RPi.GPIO, spidev, direct messages for I^{2}C, and more. This all occurs under the overarching adafruit_blinka layer. So, most codes for CircuitPython will easily be runnable on the RPi and even transferrable to other microcontrollers and Linux Computers.

The environment/API of CircuitPython would need to be installed on the Raspberry Pi first, in addition to Python itself. Then, any text editor may be used to write the programs saved as a .py file. These may then simply be run from the Command-Line Interface (CLI).

The smartphone app- developed primarily by the computer science capstone team- is in a different language suitable for mobile applications. However, this is compatible with Python: when the Bluetooth connection is established, one may run a certain .py file, or any instructions that can be run on the command-line terminal of the Pi. Taking the image is a single line command. To run any Python code, the .py file name must simply be typed into the terminal and sent.

Based on all these established frameworks, my work as described here is simplified and focused on the hardware: Creating electrical circuits within the hardware and simple python files for each module. The relevant code may be found in the appendices.

IV. Temperature Controller

A. Concept Development

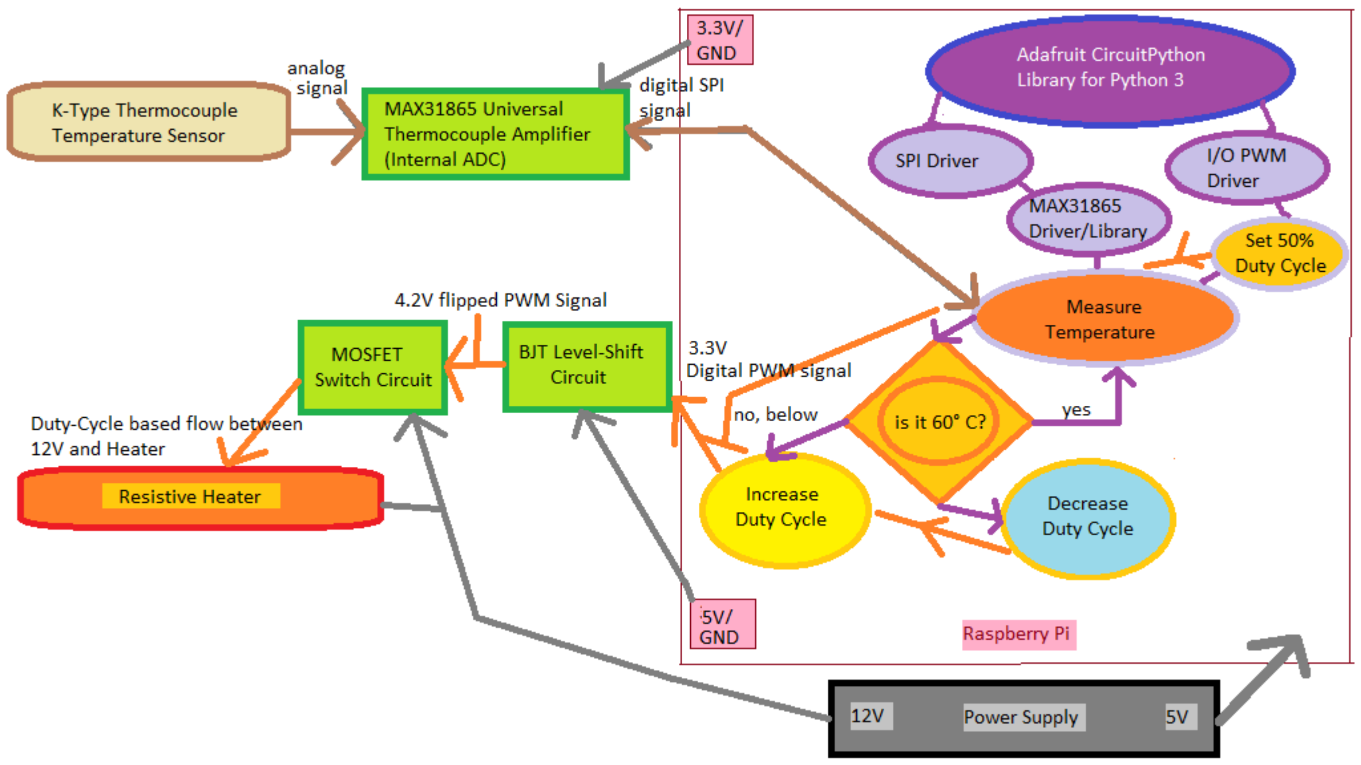


Fig. 13: Detailed Diagram of Temperature Controller

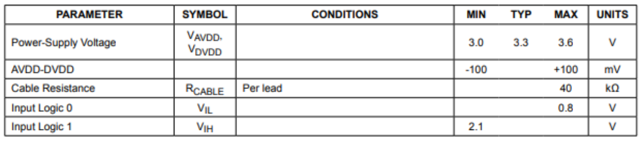
B. Implementation of Hardware

1) Thermocouple Amplifier Module

The amplifier initially in the lab was MAX6675 [18]. After attempting to write code and often running into errors where a 0 would erroneously be measured, The author found that modern software no longer supports this amplifier, and it was discontinued.

Thus we ordered the MAX31856 which has specific error codes for troubleshooting and better protection [11]. This amplifier also had a specific CircuitPython library for more ease of programming [19]. Finally, it is compatible with many types of thermocouples and not only K-type. This improves the reproducibility of the system. The table below shows us that a Pi output of 3.3 V is perfect for powering the board.

Table II: Thermocouple Amplifier Specifications [17]



PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Power-Supply Voltage	V_{AVDD}, V_{DVDD}		3.0	3.3	3.6	V
AVDD-DVDD			-100		+100	mV
Cable Resistance	R_{CABLE}	Per lead			40	kΩ
Input Logic 0	V_{IL}				0.8	V
Input Logic 1	V_{IH}		2.1			V

2) MOSFET circuit

The initial idea was to simply use the MOSFET as a switch between the 12V supply and the heater, and have it be controlled by the software. Care was taken to choose a MOSFET that had a low enough threshold voltage $V_{GS\text{-}TH}$ that the Pi's 3.3V of all programmable I/Os could control ($V_{GS\text{-}TH}$ is amount of voltage that turns the MOSFET "on" to allow the connection between the 12V and heater to properly form).

The IRLB8721 was chosen, having a minimum threshold of 1.35V-2. [10]. The component was first found on a website that particularly sold Raspberry Pi components [20], but we ended up ordering individual MOSFETS from a separate electronics supplier and it was no longer posted in the store.

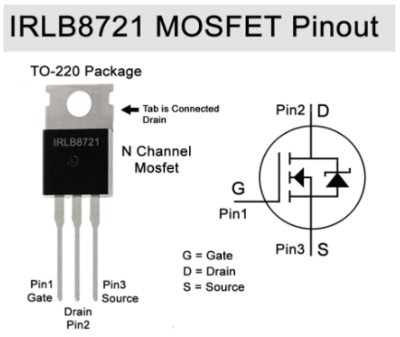


Fig. 14: IRLB8721 Pinout Diagram [9]

However, it was found through testing that since the other power source was 12V - much higher than 3.3 - it needed the max $V_{GS\text{-}TH}$ of 4V to operate. Thus, we had to create a level-shifting circuit going into the MOSFET gate.

3) BJT circuit

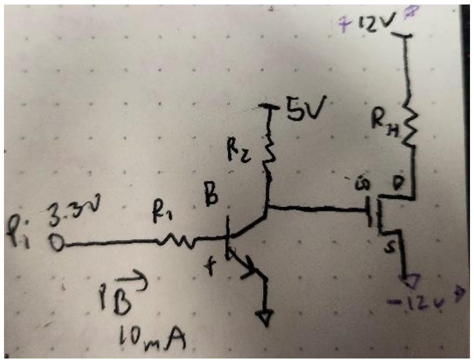


Fig. 1: Drawn Diagram of BJT Circuit

The BJT circuit is on the left, connected to the Pi and the 5V source. The elements to the right of it are the original MOSFET circuit, originally meant to be connected to the Pi pin. 3.3V is the 'high' setting, but the 'low' setting of 0V is also an option; the level is based on the programming. Many general-purpose BJTs should work for this, we had the 2N3906 on hand and used that.

a) Calculations for R1 and R2

The resistor R1 is calculated from the Pi's output voltage and current of 3.3 and 10 respectively, taking into account the voltage drop of ~0.7 occurring across the BJT.

$$R_1 = \frac{3.3\,V - 0.7\,V}{0.01\,A} = 260\,\Omega \;\rightarrow\; R_1 = 270\,\Omega$$

The minimum DC Current gain is assumed to be 50, from the 2n3906 data sheet, which gives minimum ranges from 30-100.

$$\beta\ gain = 50$$

$$I_c > 500\text{ mA} = I_B * 50$$

$$R_2 = \frac{5\,V}{0.5\,A} \geq 10\,\Omega \rightarrow R_2 = 1.8K$$

Testing the maximum gain of 300 from datasheet.

$$\beta\ gain = 300 \text{ then } I_c > 3\text{ A}$$

$$R_2 = \frac{5\,V}{3\,A} \geq 1.6\Omega \rightarrow R_2 = 1.8K$$

Setting resistor R2 as 1.8K, a much higher number that still fulfills the at-least-10, was suggested by Professor Basu.

4) Temperature Controller as a Whole

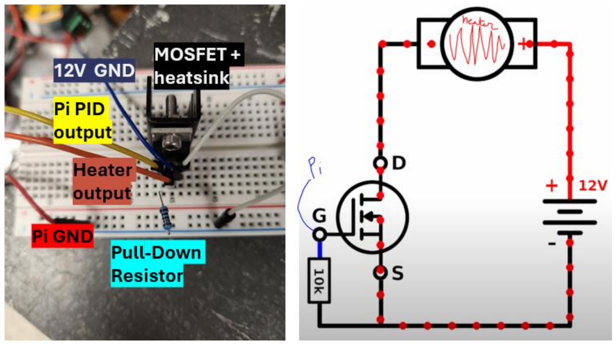


Fig. 2: Labelled Image and Diagram of Temperature Controller Prototype

This was first set up on a prototyping breadboard for the majority of the testing. We refashioned the circuit on a cut project breadboard to be small enough to fit into the handle of the final housing once it was confirmed to work from the breadboard testing.

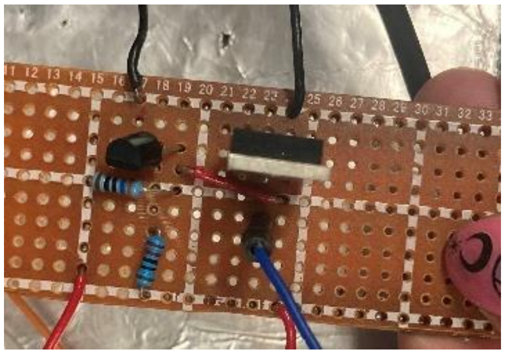


Fig. 3: Image of Temperature Control Module on Circuit Board

C. Software Implementation

The initial approach to the heating controller software was a Proportional Integral Differential (PID) control loop. This is an algorithm which uses feedback to control and automate a system and is used widely in many applications. Some of the following knowledge is from prior classes that taught the concept in MATLAB and thus remains uncited. The article by Szymon [21] was particularly helpful in translating into Python code. [22] breaks the PID concept down further.

This formula has three parameters/constants/terms that must be set, corresponding to the three words in its name. The Proportional term Kp, Integral term Ki, and derivative Kd. This parameter would be a constant, but what number is best for the controller depends on the system it is being used in. As such, it needs to be tuned to the right constants, within the system it is intended for.

Any code implementation that was discussed here will be available within the Appendices.

1) *Procedure of Tuning a PID Controller*

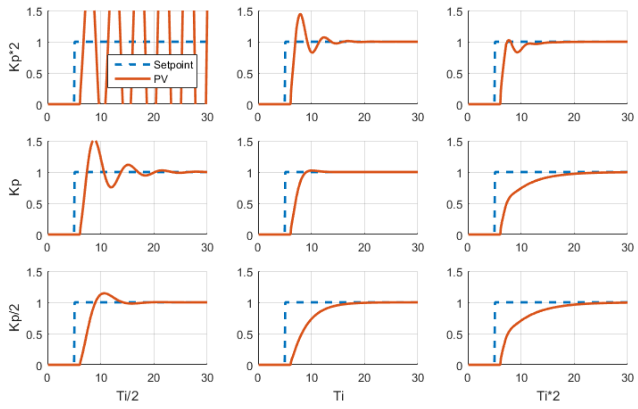


Fig. 4: PID Controller Variable Graphs [22]

This figure uses Ti for my Ki and Td for my Kd. The author will continue to use Kp/Ki/Kd in this paper.

PV stands for Process Variable. This is the measured value, and the X-axis is of time. For our needs, this would represent the measured temperature. The setpoint is what the goal value of the PV is, for us this is 60°C. The formula is as follows, *e* representing the error between the setpoint and the PV.

$$C(t) = K_P e(t) + K_I \int e(t)d\,t + K_D \frac{de}{dt}$$

This is the formula that shows clearly how the proportional, integral, and differential term show up. The following system of equations is how they are implemented in the software.

$$P = e$$
$$I = I_p + e(t - t_p)$$
$$D = \frac{e - e_p}{t - t_p}$$
$$MV = offset + K_P(P) + K_I(I) + K_D(D)$$

I was able to implement the PID closed-loop control in software, but many issues kept coming up in the circuit itself. The primary issue that took up much of the time allocated to heating was the need for the level-shifting circuit. Thus, a different angle was suggested and tested once the circuit of the heater was properly controllable from the software.

2) *Duty Cycle Testing*

The mechanism to precisely control the heat is the Duty Cycle. As such, it was theorized that a steady-state temperature might be reached by keeping the heater input at a certain percentage. Even at 100% duty cycle —i.e., 'always on'—the heater took several minutes to reach 60 °C. The precise data was not recorded but it would consistently take between 5-10 minutes to heat.

During the test of this theory, the duty cycle would be set at a level for 5 minutes to reach an approximation of the steady-state temperature in an open-loop program. This python script increments the percentage by 16 divisions. This number was the easiest between 5% and 10% to implement in a binary number. The temperature is measured every minute, and the cycle percentage is incremented every 5 minutes. The initial test was completed with the heater placed into insulating Styrofoam to get data uninfluenced by fluctuations of the external environment.

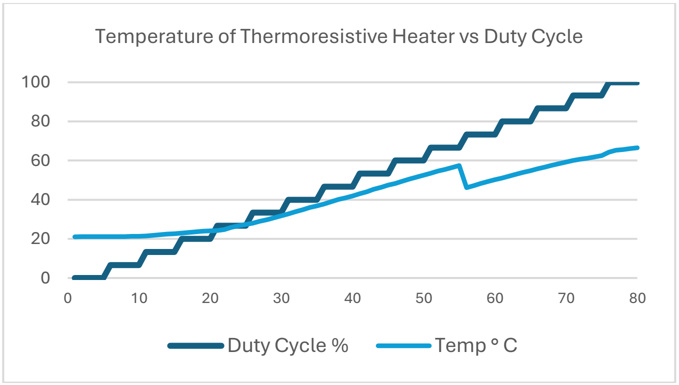


Fig. 5: Graph of Temperature Vs Duty Cycle Experiment 1

There was an error at 66% that caused the program to turn off which was noticed about 5 minutes later before restart. The error was addressed and the program restarted at 53% hoping it would reach near the same temperature once back to 66%. As seen here, there was still a lower temperature by that point. Due to time constraints the entire test could not be redone.

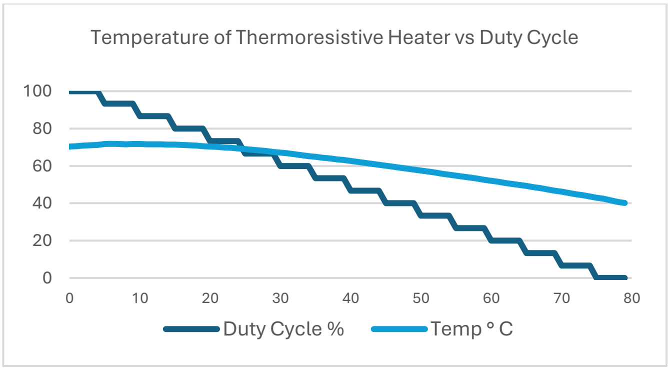


Fig. 6: Graph of Temperature Vs Duty Cycle Experiment 2

The next graph shows the same experiment starting at 100% and decrementing. The combination of data from incrementing and decrementing shows the steady state range.

The conclusion that can be drawn from these graphs is 46.67% to 80% is the range of duty cycles where the system may reach steady state at 60 °C. Further testing with; smaller increments and a time range of 10 minutes is recommended to find a proper steady state. Further testing was done in a prototype housing that had more holes to the external environment than the final housing would. There was not adequate precise data collected to display and draw a specific conclusion, but generally the associated temperature for each level of duty cycle was 10-15% lower than in the properly insulated environment.

V. LED CONTROLLER

A. Concept Development

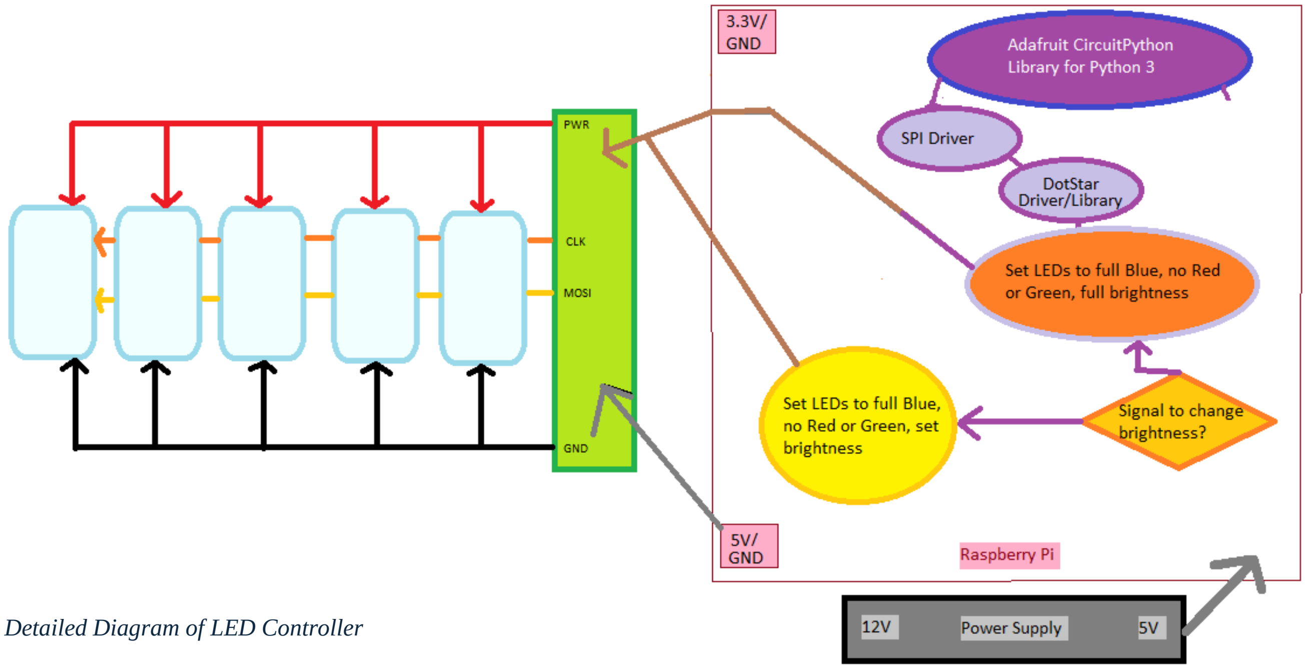

3.3V/ GND
Adafruit CircuitPython Library for Python 3
SPI Driver
DotStar Driver/Library
Set LEDs to full Blue, no Red or Green, full brightness
PWR
CLK
MOSI
GND
Set LEDs to full Blue, no Red or Green, set brightness
Signal to change brightness?
5V/ GND
Raspberry Pi
12V
Power Supply
5V

Fig. 15: Detailed Diagram of LED Controller

The basic requirement for the LED was to be at a certain wavelength and take good images of the chip in low-light environments. The figure below shows a test we ran in the initial stages with a basic LED at different voltages and amperages from a prototyping lab power supply.

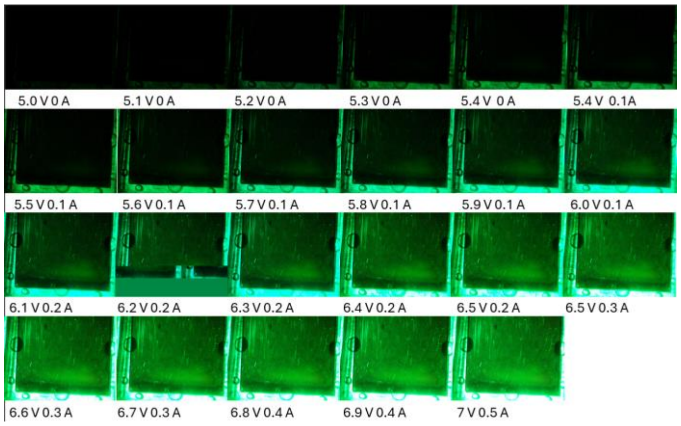


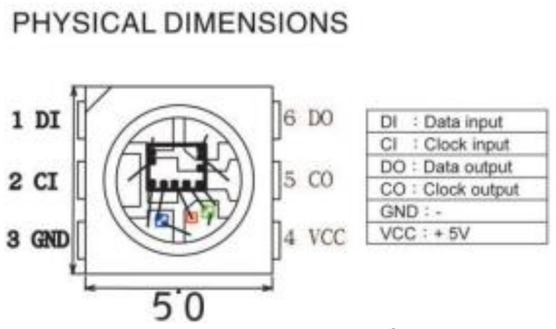
Fig. 7: LED power test

These results show the need for a high level of power for clear images. An alternative considered was using another switch circuit similar to the heater, but there are many low energy LED options in the market. Between the results of this test and the larger and awkward form factor of those LEDs, we moved on to find a better solution.

I decided to develop a ring light PCB to provide even lighting. To acquire the proper images, the fluorescent solution developed by Dr. Ram's team needs to be excited by a light at a certain wavelength. The specification given is a wavelength between 450-470 nm.

We found a mini, multipurpose, and addressable LED component based on these specifications – the SK9822, or "DotStar," by Adafruit [5]. These LEDs met the wavelength and form-factor requirements [5]. Both these LEDs and the MAX31865 amplifier use SPI communication, which simplifies integration.

These components even had the ability to communicate through SPI and control the color and brightness through code, rather than needing to adjust the power manually to change brightness. The following figures are parts of the datasheet for said component [5].



DI	: Data input
CI	: Clock input
DO	: Data output
CO	: Clock output
GND	: -
VCC	: + 5V

Fig. 8: Physical Dimension Diagram of Dotstar LED

The small size of 5 mm, SPI compatibility, and 5V power allows it to work well with the existing Pi system.

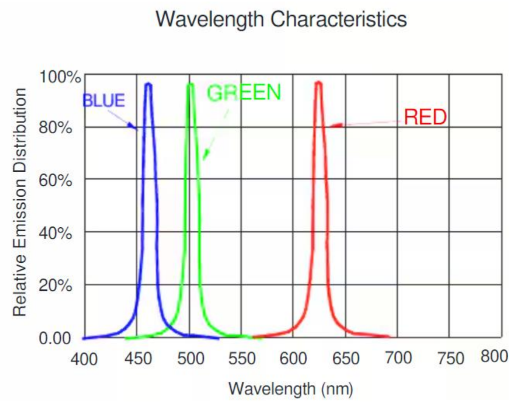


Fig. 9: Wavelength Graph of Dotstar LEDs

This was prototyped on a breadboard as shown below to test brightness and color changeability through the software.

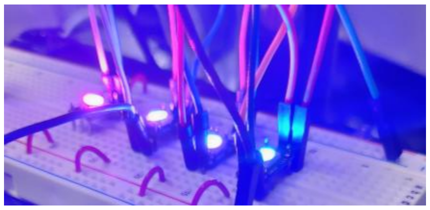


Fig. 10: Image of LED Prototyping- Changing Color

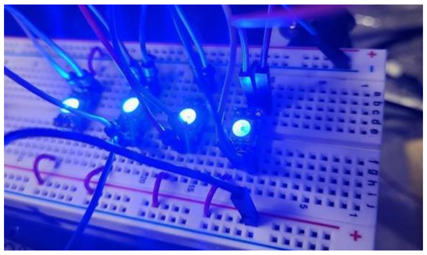


Fig. 11: Image of LED Prototyping- Changing Color

1) PCB Development

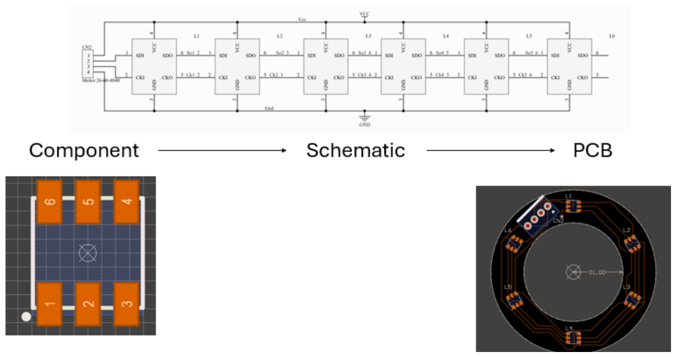


Fig. 12: PCB Development Overview

This figure shows the overall process of developing a PCB that worked with the LEDs. The author used the software CircuitMaker developed by Altium [23]. The following figures include closer and detailed screenshots.

First, a model of the component itself is necessary. This one was found in the library of the design software when searching for the component sk9822.

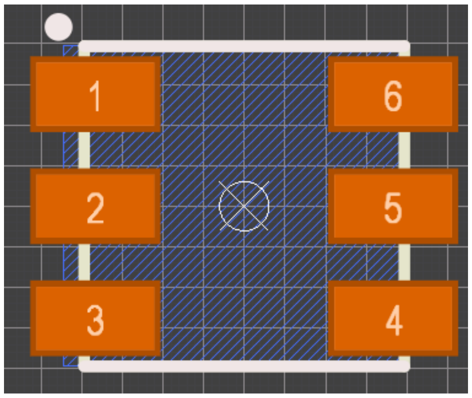


Fig. 13: DotStar LED Compoment Design

2) Schematic and PCB layout of LED controller

Using that component, the author created a schematic within the system for how many LEDs we would like to use for even illumination and proper connections. This is shown on the right column.

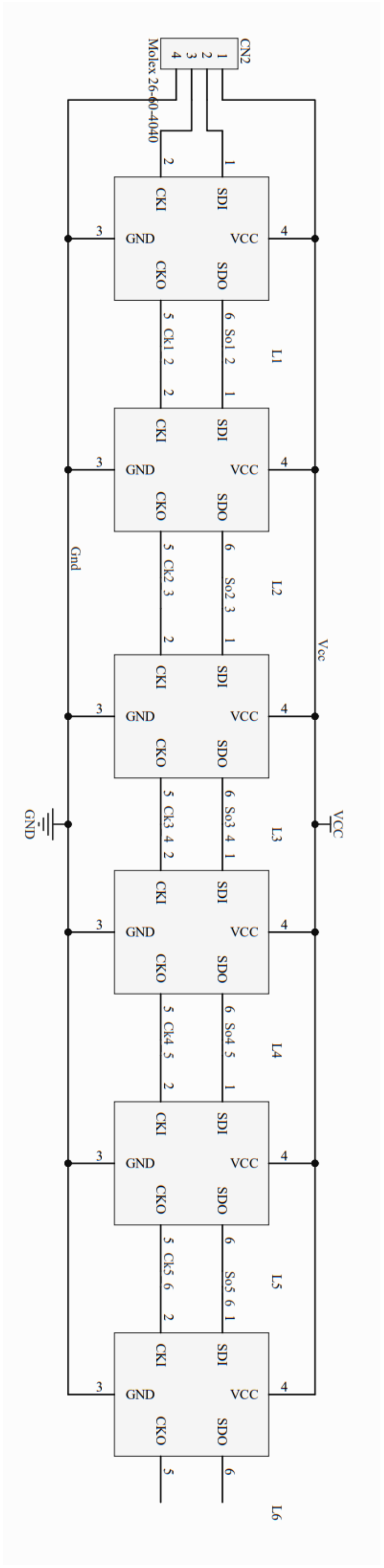


Fig. 14: Schematic of LED Ring Design

The design is then created on the PCB itself in the software (Altium CircuitMaker).

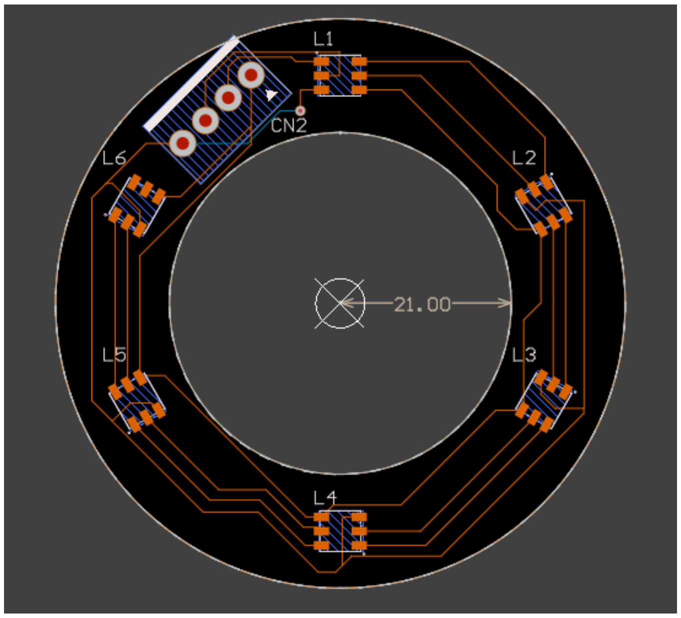


Fig. 15: Initial LED PCB Design

The schematic is used by the software to do a Design Rule Check (DRC) comparing the design to ensure it fits the schematic. The DRC also compares the design to general industry standards and constraints that are included in CircuitMaker. The design was also checked by Basu, and then ordered through Oshapark.

B. *Troubleshooting faulty PCB*

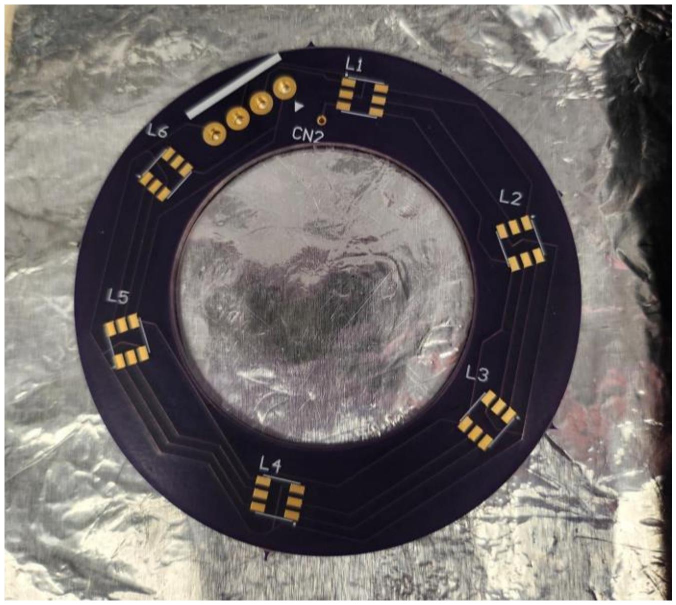


Fig. 16: Manufactured Ring Light PCB

The first print of the PCB came in, and the LEDs were soldered on. Following is the image of the housing with the board placed within, as also shown in section II-D.

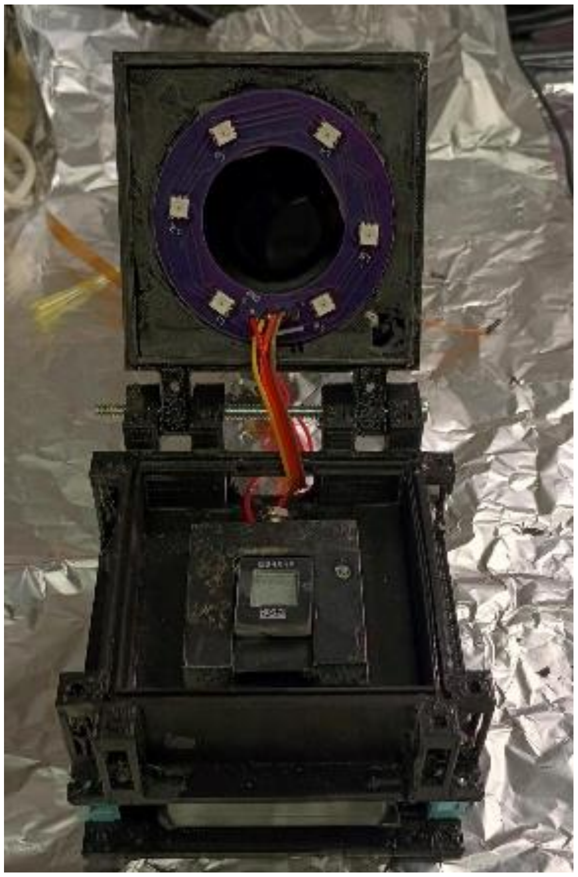


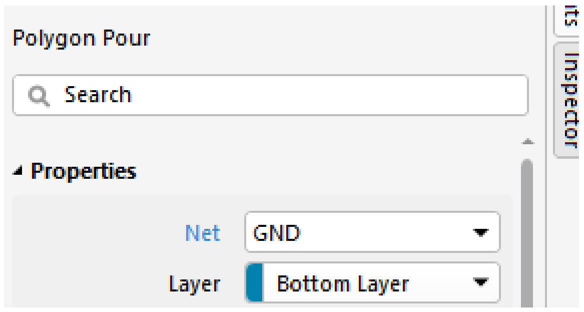
Fig. 16: Manufactured Ring Light PCB in Housing Prototype

It turned out that this PCB did not work for more than 2 LEDs, sometimes not even 1 LED. There were a few printed and three of them were tested. Then the original design had to be revisited when realizing that the original PCB design lacked a ground plane- a standard design feature for circuit boards throughout the industry.

C. *Finalizing the PCB*

The PCB was then revisited in CircuitMaker. The VCC trace was made thicker, and the grounds all passed through a via near the ground pin of each LED. The bottom plane of the PCB was made basically into a big ground that was connected with the ground header pin and each via.

This was done through the Polygon Pour function, which allows you to select one of the nets from the schematic and select a region and a layer to 'pour,' allowing spaces for the other components.


Polygon Pour

Search

Properties

Net: GND

Layer: Bottom Layer

Fig. 17: Screenshot Showing the CircuitMaker Polygon Pour Tool

Here is the updated top layer of the PCB first.

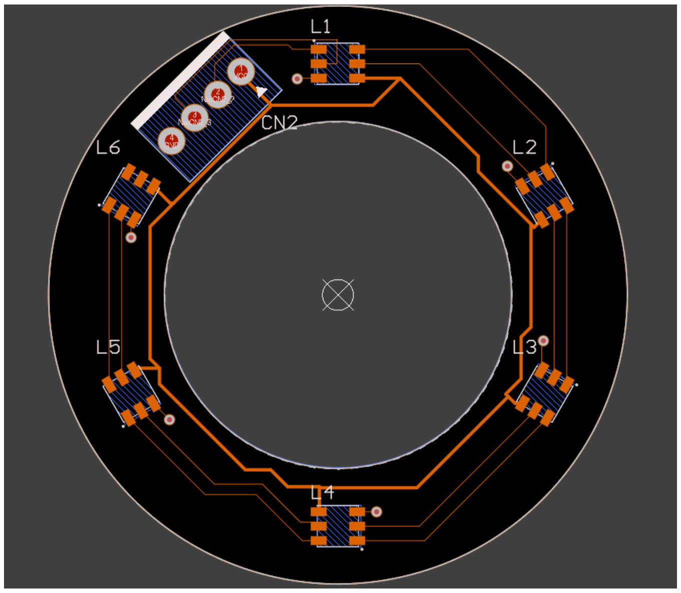


Fig. 18: Updated LED PCB Design Top Layer

Here is the bottom layer of the updated PCB, and then a close-up showing how the non-ground vias are blocked off.

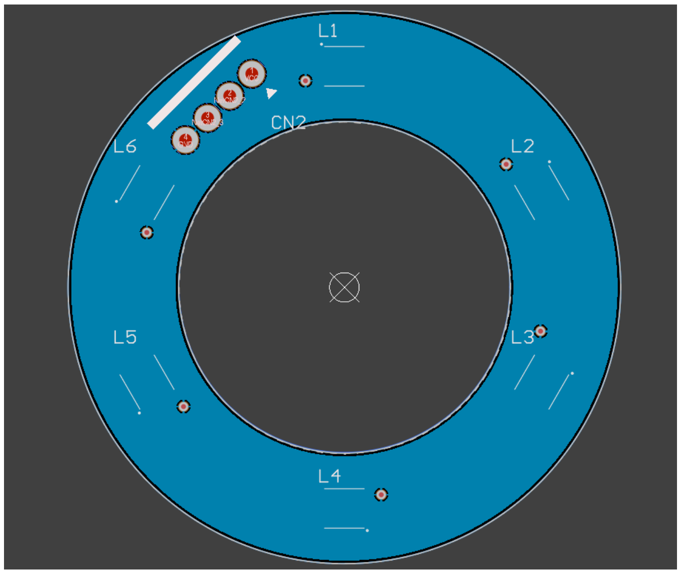


Fig. 19: Updated LED PCB Design Bottom Layer

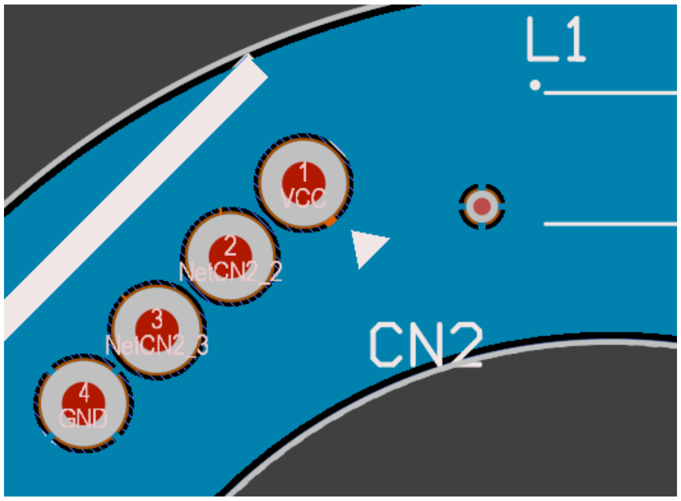


Fig. 20: Updated LED PCB Design Bottom Layer- Zoomed In

We then ordered the updated PCB and some additional LEDs, soldered them on when they came in, and it worked properly.

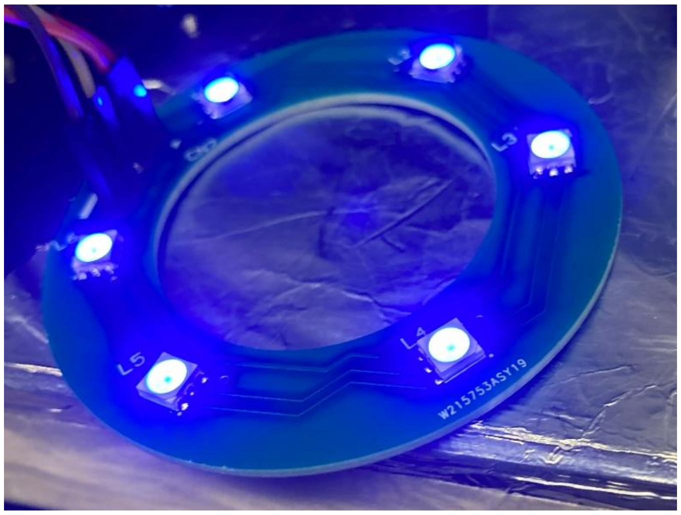


Fig. 21: Manufactured Ring Light PCB with Working LEDs

D. *Software Implementation*

These LEDs use SPI and are sold on Adafruit just like the thermocouple amplifier. So, implementing the software was very similar to the amplifier. CircuitPython has a Dotstar library as well [24]. [25] includes detailed example code that was primarily targeted for the use-case of a decorative LED strip. Our application was much simpler from the code side, with a few LEDs meant to stay one color, with changeable brightness.

There was some experimentation on writing a script that manually sent out the SPI frame necessary for the LEDs based on the datasheet. This did have success, but ended up as a somewhat bulky code. This would be solvable by condensing some of the code into sub-functions in headers of my own. Once the implementation on CircuitPython was found, this was no longer necessary. Both versions are shown in the appendices.

VI. Imaging

A. *Tuning the Camera Settings*

For imaging, the process was quite simple. The Raspberry Pi OS comes pre-installed with some camera functionality [26]. This command-line controlled app enables tuning the camera settings for the best images in the necessary lighting conditions. The goal is to get the image with the brightest and most distinct excited positive samples (or fluorescein to emulate fluorescent samples) on the chip in low lighting.

I found that the best ISO was the max of 800. The ISO is one of three camera settings that affect the exposure in an image. The other two include aperture, or the size of the hole in the lens that allows the light through, and the shutter speed, or the length of time the shutter is open and the sensor is exposed to light [27]. The ISO is set using the '-iso' option in the command line, and the range through this software is 100-800.

A lens was selected based on its performance for close objects, and the sensor was proven to behave well in low lighting, but the software is not able to control the aperture. The other setting we have control of is the shutter speed, indicated by the '-ss' option in the command line. The units used for this command are in microseconds, so 1,000 is one millisecond and 1,000,000 is one second.

I used the video mode of the application, Raspivid, to determine the focal distance necessary to have the best clarity of the chip. This measurement was used to inform housing design. Raspivid was also used as a sort of preview while loading the chip with fluorescein. The following figures show the process of narrowing down a good shutter speed and are not inclusive of all the test pictures taken. Lower ISO and shutter speed settings resulted in dark images. A too-long shutter speed would wash the sensor out resulting in bright washed-out images.

1) *Imaging Fluorescein droplets using different exposure settings*

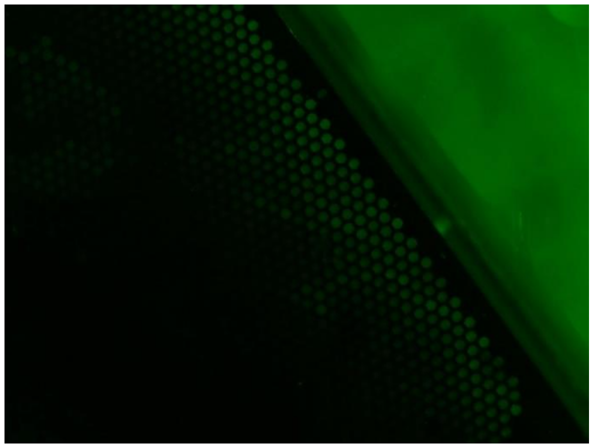


Fig. 22: Fluorescein on chip imaging test for ISO 800 SS 500,000

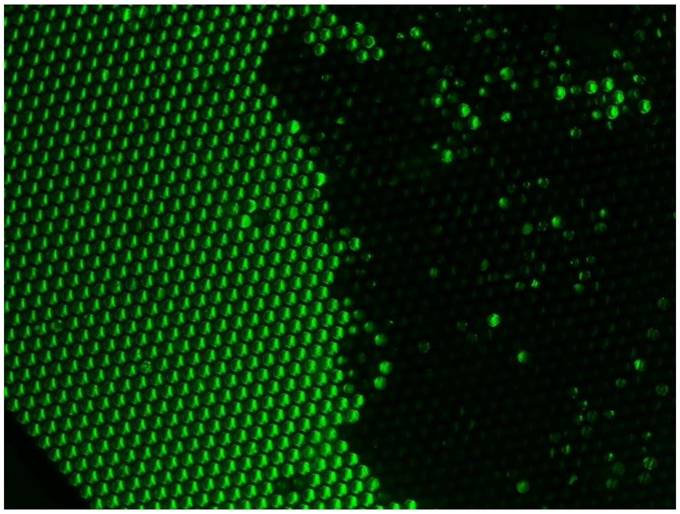


Fig. 23: Fluorescein on chip imaging test for ISO 800 SS 1,000,000

Seen in Figs. 16 and 17, the 1-second time shows the cell wells distinctly and the differences in cells with and without fluorescence. However, there is some blurriness in part of the image with 1 second.

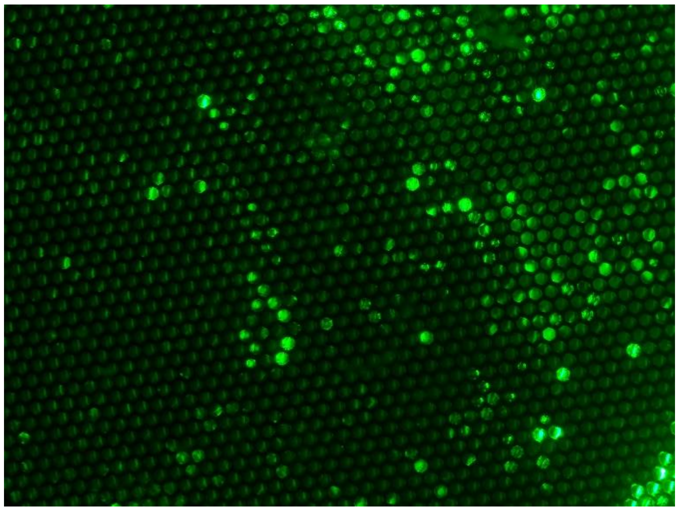


Fig. 24: Fluorescein on chip imaging test for ISO 800 SS 3,000,000

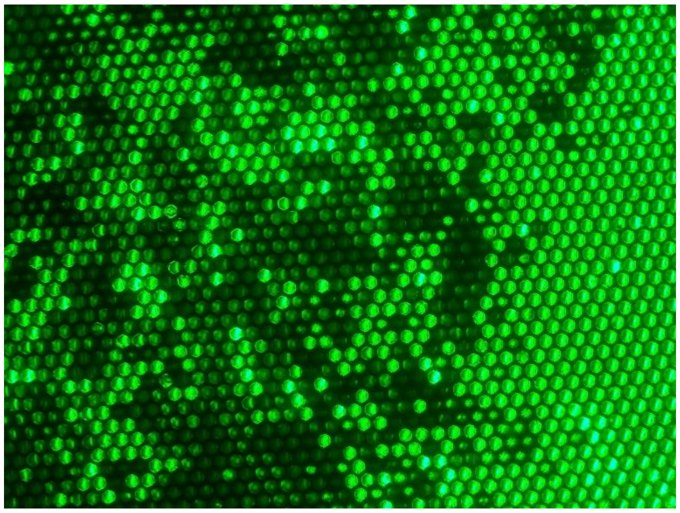


Fig. 25: Fluorescein on chip imaging test for ISO 800 SS 4,000,000

In Figs. 18 and 19, the brightness washes out the image a bit more and the cells that have limited to no fluorescein are still somewhat illuminated. There is still good clarity of the image and chip itself, but the differences between the cells are harder to make out. Thus, the shutter speeds used for images of best clarity ranged from 1-4 seconds. The longer times in this range give a clearer and brighter result. 3 seconds is ideal for true low lighting as this system will have within the final housing. A time that was too long would wash the sensor out and could be too bright or blurry. The prototype allowed some limited environmental interference.

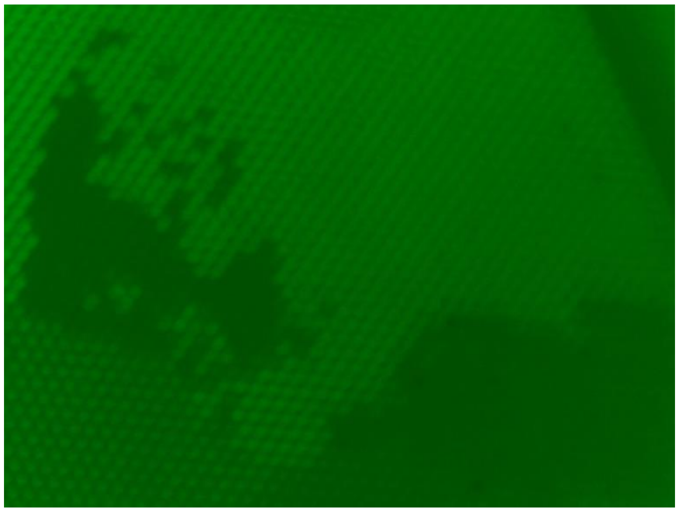


Fig. 26: Fluorescein on chip imaging test ISO 700 SS 5,000,000

Fig. 20 shows the washed-out blurry effect of a higher shutter speed and lower ISO.

The following images are some more examples, all taken through this system with the same settings as described in the caption of Fig. 21.

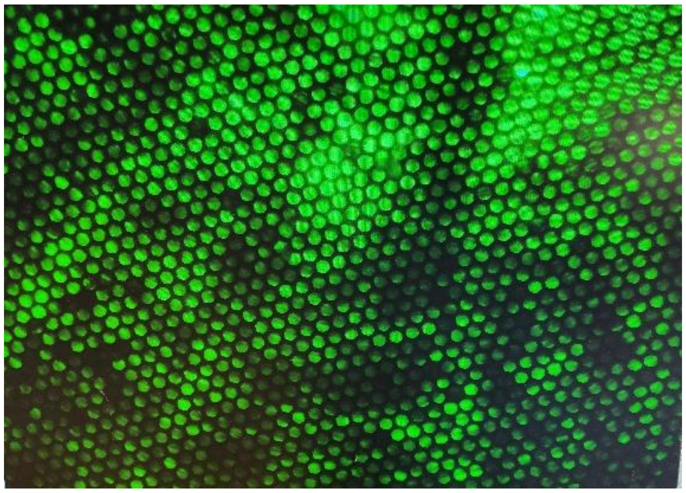


Fig. 27:Fluorescein on chip imaging test for ISO 800 SS 3,000,000

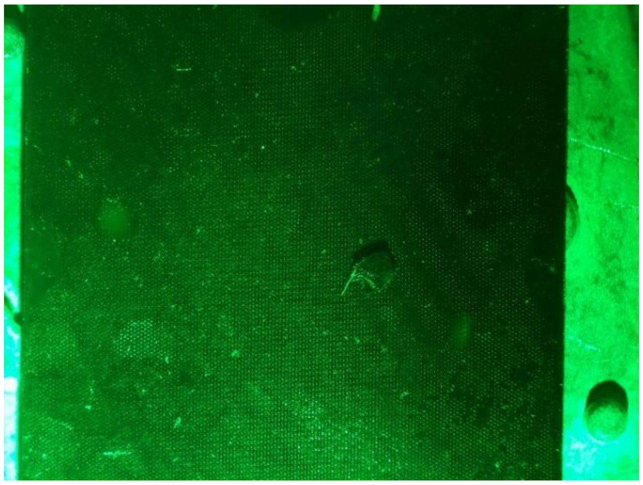


Fig. 28: Image of empty cell-well chip

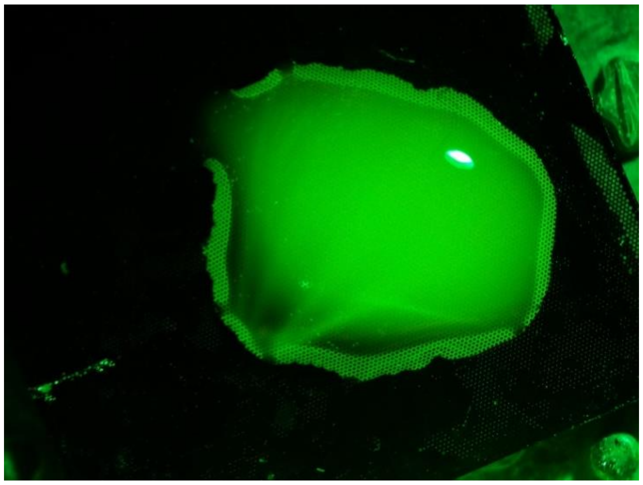


Fig. 29: Image of Fluorescein droplet initially applied to chip

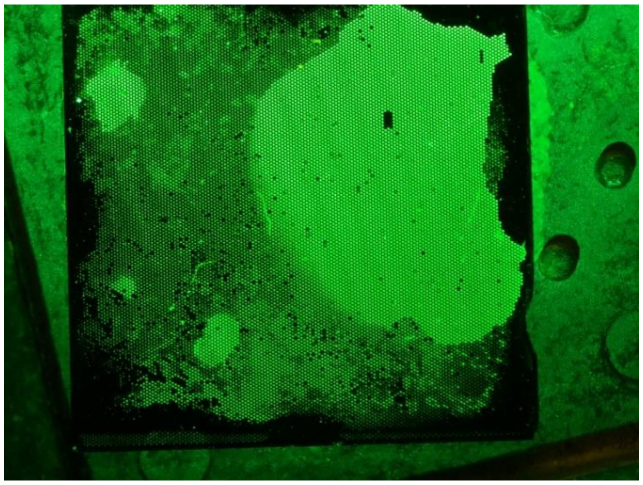


Fig. 30: Image of Fluorescein droplet spreading on chip

Figs. 23-25 were taken to understand how a drop of liquid would spread on a chip, and to determine the focal distance and positioning to include an entire chip in the image. The previous images were closer to about a fourth of the chip. The proper loading process of the chips includes using a squeegee to evenly distribute a sample's liquid, and then application of a lid and immersion fluid to properly isolate droplets.

B. *Software Implementation*

As discussed in the overall design description, the smartphone app is designed to run Python codes on the Pi when buttons on the app are pressed. The standard Python module 'Subprocess' provides functionality to run any kind of command-line function through Python code [28]. Thereby the RaspiCam CLI app used to determine the proper camera settings was easy to convert into python code. The sources [29] and [30] have examples and influenced the code.

Similar to the other modules, the code is available in the Appendices. In the mobile app there is functionality just for taking the image, just for adjusting or testing the lights, and other functionality to run the full test. The option to run the full test implies a code that stacks each function. The appendices separate the code by the specific module and functionality that is necessary. For running the full test, one would combine the code so it would set up all the hardware variables, light up the LEDs to set brightness, and run one big loop that contains simultaneous imaging and temperature control. Particularly important is the temperature being maintained throughout and between the imaging process of 60 minutes with a picture each minute.

Within the imaging code, the images are labelled with their timestamp. So, this can be checked against the previous timestamp to ensure that about a minute has passed. Alternatively, in the PID controller code, there is a loop which uses time.sleep(x) (which takes the input x in seconds) to delay each loop. The built-in RaspiCam timelapse function, also shown in the imaging code, offers another option for periodic imaging. The other functions of the test could thus run parallel to the image capture timelapse.

VII. ddLAMP EXPERIMENTAL RESULTS

A. *Sample Testing*

Although the full working system was not able to be tested, the imaging sensor's functionality in this was. A collaborator from the Chemistry team conducted a sample test at a midpoint of both teams' development. The process includes heating a positive and negative/control sample, placing each on its own chip, and imaging under the necessary lighting.

We tested the samples under 3 different exposure settings, both in their tubes and on the cell-well chip. On the engineering side, we could then see what setting gives the best result. On the chemistry side, the testing was to figure out if the negative and positive samples were distinct and accurate enough, or if more tweaks needed to be made to the formula. The ISO was the same, 800, for all of them. The captions of each figure include the shutter speed setting.

B. *Imaging sample droplets*

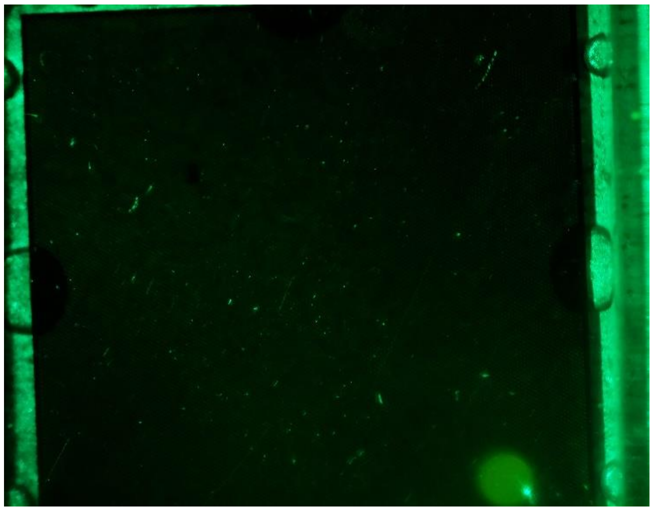


Fig. 31: Negative Chip 1 SS 1,000,000

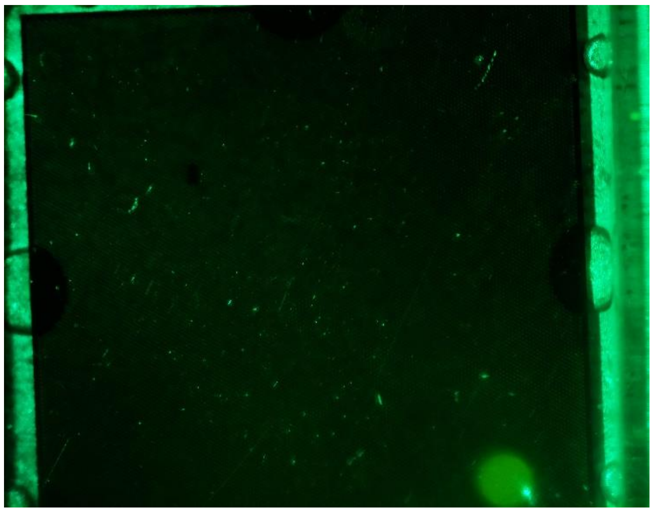


Fig. 32: Negative chip 2 SS 2,000,000

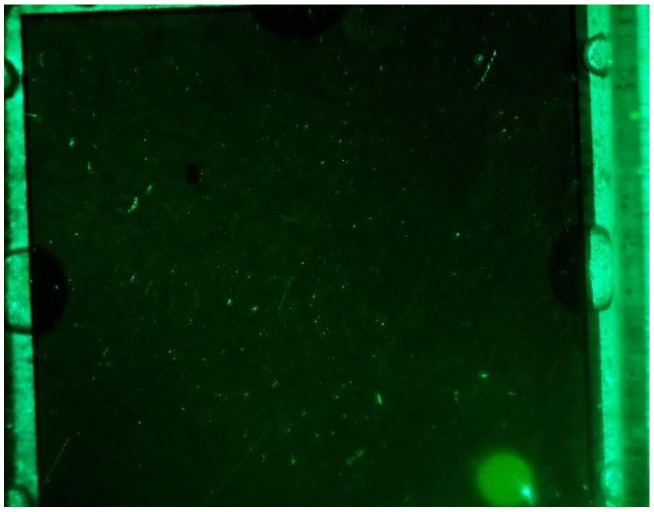


Fig. 33: Negative Chip 3 SS 3,000,000

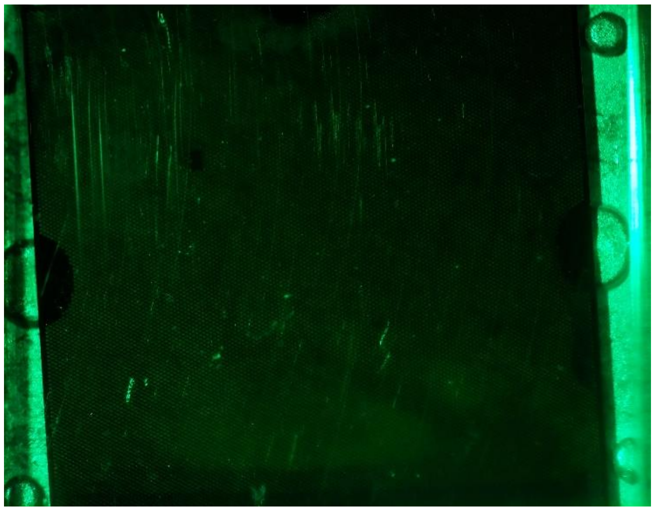


Fig. 34: Positive Chip 1 SS 1,000,000

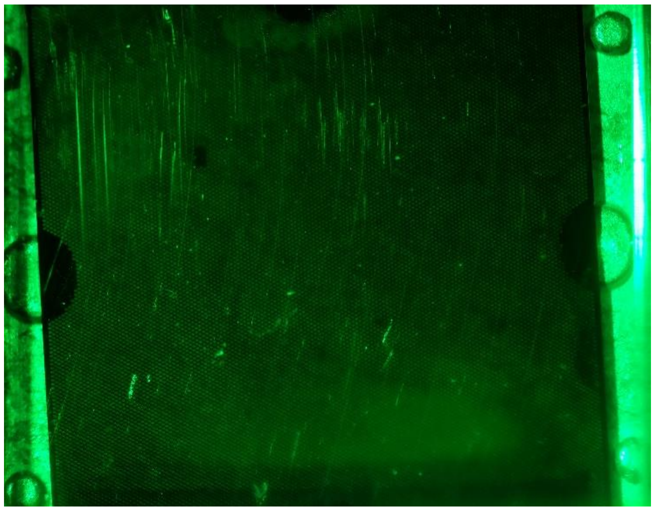


Fig. 35: Positive Chip 2 SS 2,000,000

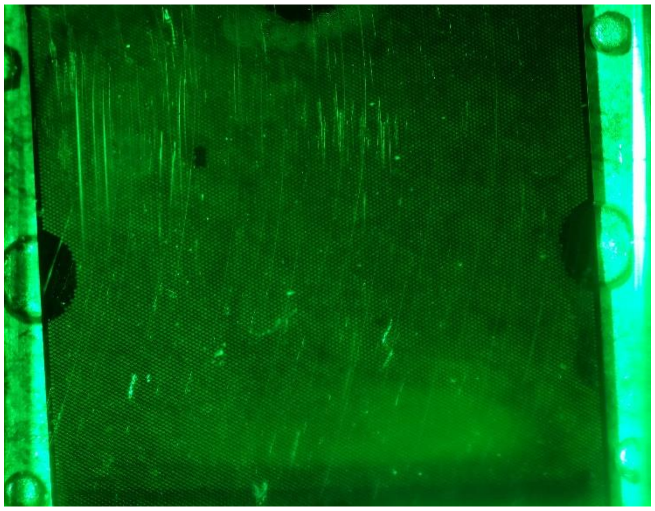


Fig. 36: Positive Chip 3 SS 3,000,000

The positive chips' images appear to be brighter overall and not just the liquid, looking at the reflectiveness of the aluminum backing. This is most probably a difference in lighting adjusted between removing the negative and inserting the positive chip. The test occurred prior to ring light integration, resulting in uneven light. Partially because of this, and partially to see the difference in fluorescence at a higher concentration, another reaction was prepared and images

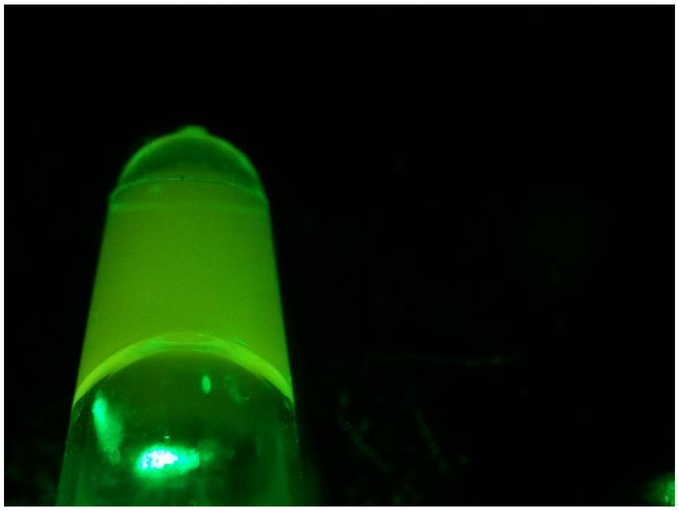


Fig. 37: Negative Sample 1 SS 3,000,000

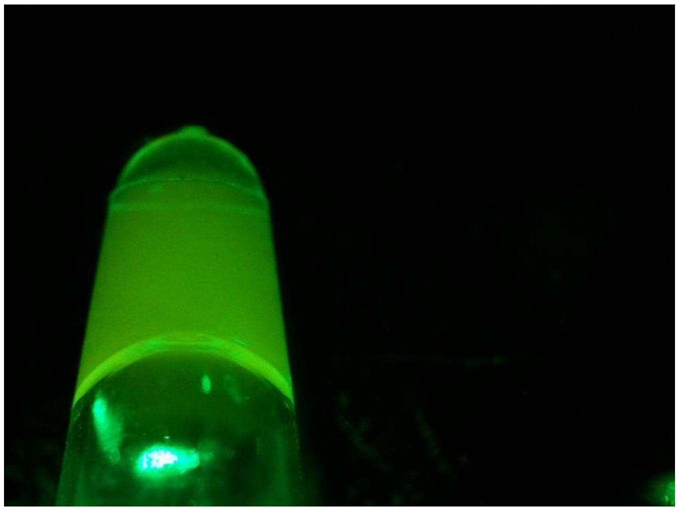


Fig. 38: Negative Sample 2 SS 2,000,000

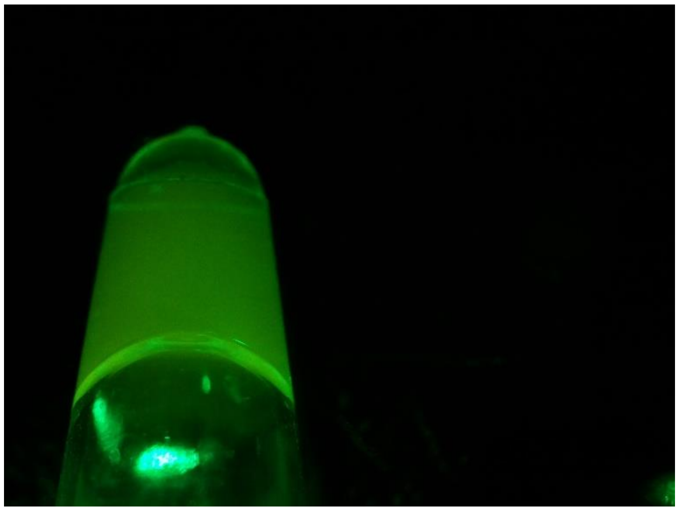


Fig. 39: Negative Sample 3 SS 1,000,000

of the liquid in these tubes were also taken. The samples were first heated for the necessary time, about 60 minutes. These images were taken afterward.

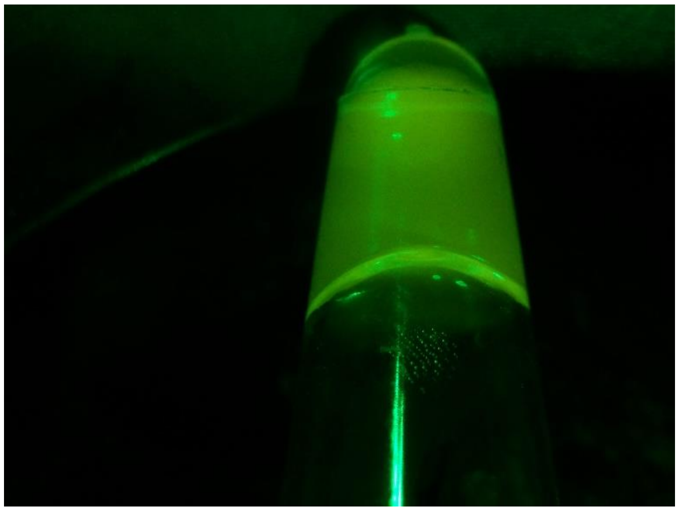


Fig. 40: Positive Sample 1 SS 3,000,000

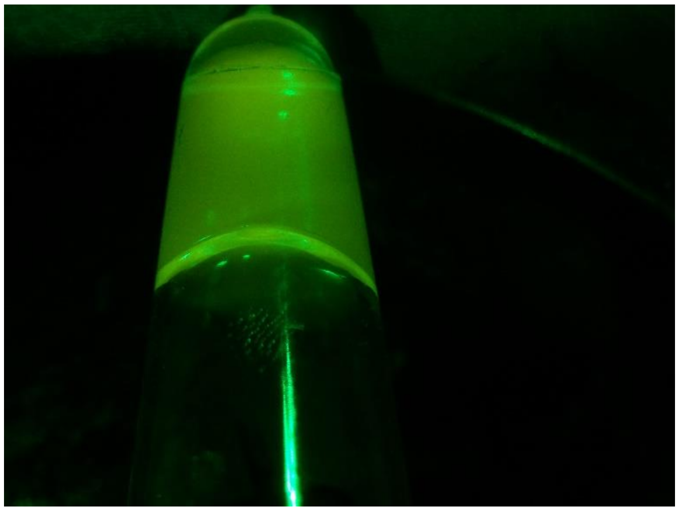


Fig. 41: Positive Sample 2 SS 2,000,000

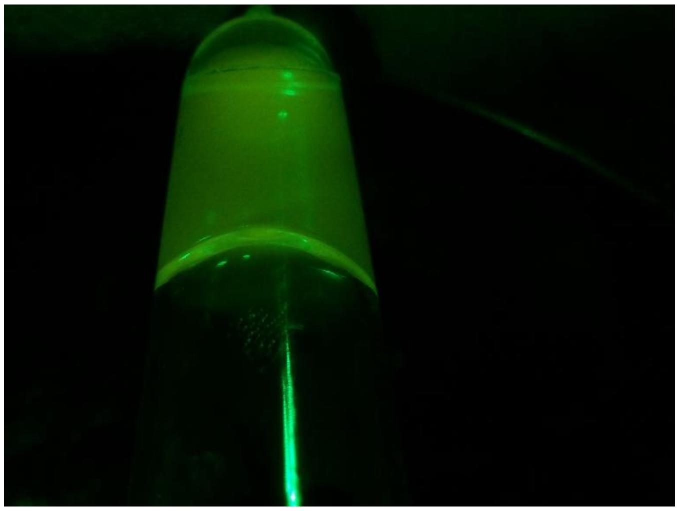


Fig. 42: Positive Sample 3 SS 1,000,000

These results show differences between the positive and negative samples, and the images are more consistent. More work needed to be done on the compound to create a larger distinction in fluorescence according to the chemistry team.

Due to the timing of this test, the finalized heating and lighting modules were not yet integrated. A second test with the full system remains necessary to determine the viability of the complete prototype and assess the fluorescent's compound performance.

VIII. Conclusion

This thesis documents the design and development of a portable ddLAMP imaging system that integrates hardware and software for microbial source tracking. Each subsystem was developed and tested independently. While the imaging platform performed effectively in preliminary tests, a fully integrated test with finalized heating, lighting, and imaging modules remains pending.

Further testing with optimized hardware and a further refined fluorescent compound is required for evaluation and validation of the system. Nevertheless, the design and functionality of this prototype demonstrate a strong foundation for future development and field deployment.

Appendix

IX. Code for Imaging

```
# takePic.py
# Sana Hasan
# Jan 2025

import time
import subprocess

print ("Taking picture")
# single picture timestamp version
tmStmp = time.strftime("%d-%m-%Y_%H:%M:%S")
cmd = "raspistill -o Desktop/" + tmStmp + ".jpg"
# timelapse version - 1 picture a minute over 60 minutes, making its own timestamped directory per test/run
cmd = "mkdir test" + tmStmp
cmd = "raspistill --timeout 3600000 --timelapse 60000 -o test" + tmStmp + "/image%04d.jpg -ISO 800 -ss 3000000"
# error checking
print(cmd)
ans = subprocess.call(cmd, shell = True)
if ans == 0
      print("Image Captured")
else:
      print("Command failed.", return_code)

```

X. Code for LED lighting

```
# cp_led.py
# Sana Hasan
# May 2024

import board
import adafruit_dotstar


ledpower = digitalio.DigitalInOut(board.D26)
ledpower.direction = digitalio.Direction.OUTPUT
ledpower.value = True
brightVal = 0.7
# 70% brightness preset.
# brightness setting goes from 0.0 off to 1.0 full

while True:
    #loop initializes the LEDs and then checks if brightness must be changed

    pixels = adafruit_dotstar.DotStar(board.SCK_1, board.MOSI_1, 6, brightness=brightVal, auto_write=True)

    pixels.fill((0,255,0))
    pixels.show()

    print ("Would you like to change the brightness? \n Enter + to increase, - to decrease, or b to manually set. \n")
    changeBright = input().lower()
    #if input is + increase, - decrease by 0.1
    if changeBright == '+' and brightVal < 1.0
        brightVal += 0.1
    elif changeBright == '-' and brightVal > 0.0
        brightVal -= 0.1
    if changeBright == 'b'
        #if b then manually set
        print("Enter a value between 0.0 and 1.0: \n")
        tempBrightVal=float(input())
        if 0.0 < tempBrightVal < 1.0
            brightVal = tempBrightVal
        else:
            print("invalid")
    else:
        brightVal=0.7

```

XI. Code for PID Temperature Controller

```python
# PIDtest.py
# Sana Hasan
# June 2024

import time
import board
import pwmio
import digitalio
import adafruit_max31856


#PID SETUP


# Using the PID function, the value of MV impacts the heater
# It can be being turned on or off or the level of heat can be changed.
# The controller needs to be tuned via constants for Kpr, Kin, Kde.
# This is done through strategic trial and error

#PID variables
timepid = 0
integral = 0
D = 0
time_prev = -1e-6
e_prev = 0
#set max and minimum of PID output
out_min = 0000
out_max = 0xffff #65535 #0xffff
def PID(Kp, Ki, Kd, setpoint, measurement):
    global timepid, integral, time_prev, e_prev, out_min, out_max
    # Value of offset - when the error is equal zero (unnecessary for this)
    offset = 0
    # PID calculations
    timepid = time.time()
    e = setpoint - measurement
    #the difference between
    #measurement- measured temperature of the system
    #setpoint- what we want the temperature to be
    P = Kp*e
    integral = integral + e*(timepid - time_prev)
    D = (e - e_prev)/(timepid - time_prev)
    # calculate manipulated variable - MV
    MV = offset + Kp*e + Ki*integral + Kd*D
    # update stored data for next iteration
    e_prev = e
    time_prev = timepid

    MV=int(MV)

    #constrain output and return
    if MV > (out_max):
        MV = out_max
    if MV < (out_min):
        MV = out_min
    return MV

#create set temperature point and constants
desiredt = 60.00
Kpr = 20000
Kin = 0
Kde = 0


# HARDWARE SETUP


#Heater switch pin creation
#heater = digitalio.DigitalInOut(board.D13)
#heater.direction = digitalio.Direction.OUTPUT
#Using PWM now instead

#Pulse Width Modulation pin creation
heater = pwmio.PWMOut(board.D12, frequency=100)
heater.duty_cycle = 0xffff
#duty cycle is a 16 bit int
#where 0xffff always high and 0x000 always low

#deinit()
#deinitializes all hardware resources if needed

# Sensor object creation,
# communicating over the board's default SPI bus
spi = board.SPI()

# allocate a CS pin and set the direction
cs = digitalio.DigitalInOut(board.D25)
cs.direction = digitalio.Direction.OUTPUT

# create a thermocouple object with the above
thermocouple = adafruit_max31856.MAX31856(spi, cs)


# MAIN CODE LOOP


while True:
    # measure the temperature! (takes approx 160ms)
    currenttemp = thermocouple.temperature
    print((currenttemp,))  # in Celsius
    #pid equation function
    pwmvl = PID(Kpr, Kin, Kde, desiredt, currenttemp)
    heater.duty_cycle = pwmvl
    print(pwmvl)
    time.sleep(1) #sleep for a second

# alternative (non-blocking) way to get temperature
#thermocouple.initiate_one_shot_measurement()
# <perform other tasks>
# now wait for measurement to complete
#while thermocouple.oneshot_pending:
#  pass
#print(thermocouple.unpack_temperature())

```

XII. Code for Temperature Controller Duty Cycle Test

```python
1. # DutyCycleTempTest.py
2. # Sana Hasan
3. # Dec 2024
4.
5.
6. import time
7. import board
8. import pwmio
9. import digitalio
10. import adafruit_max31856
11.
12. # Create sensor object, communicating over the board's default SPI bus
13. spi = board.SPI()
14.
15. # allocate a CS pin and set the direction
16. cs = digitalio.DigitalInOut(board.D25)
17. cs.direction = digitalio.Direction.OUTPUT
18.
19. # create a thermocouple object with the above
20. thermocouple = adafruit_max31856.MAX31856(spi, cs)
21.
22. # create heater switch pi
23. heater = pwmio.PWMOut(board.D12, frequency=100)
24.
25. #duty cycle is a 16 bit int, this 2**15 is half
26. #where 0xffff always high and 0x000 always low
27. heater.duty_cycle = 0xffff
28.
29. while heater.duty_cycle >= 0x0000:
30.     for i in range(5):
31.         # measure the temperature! (takes approx 160ms)
32.         currenttemp = thermocouple.temperature
33.         print((currenttemp,))  # in Celsius
34.         time.sleep(60)
35.     if heater.duty_cycle > 0x0000:
36.         heater.duty_cycle-=0x1111
37.         print("moving duty cycle to", heater.duty_cycle)
38.     else:
39.         print("done!!! or error?")
40.         break
41.     #this should increment it 16 times from always on, 0xFFFF, to always off, 0x0000. For the inverse, initial cycle should be set to “ 0x0000 ” on line 27, the inequality on line 29 should be “… <= 0xFFFF”, the inequality in line 35 should be “… < 0xFFFF”, and line 36 should use += instead of -=
```

XIII. Code for Running at a Constant Duty Cycle

```python
# deinit.py
# initializer/deinitializer that sets at a constant duty cycle
# Sana Hasan
# Sept 2024


import time
import board
import pwmio
import digitalio
import adafruit_max31856

# Create sensor object, communicating over the board's default SPI bus
spi = board.SPI()

# allocate a CS pin and set the direction
cs = digitalio.DigitalInOut(board.D25)
cs.direction = digitalio.Direction.OUTPUT

# create a thermocouple object with the above
thermocouple = adafruit_max31856.MAX31856(spi, cs)

# create heater switch pi
heater = pwmio.PWMOut(board.D13, frequency=100)

#duty cycle is a 16 bit int, this 2**15 is half
#where 0xffff always high and 0x000 always low


while True:
    heater.duty_cycle = 0x0000
    # measure the temperature! (takes approx 160ms)
    currenttemp = thermocouple.temperature
    print((currenttemp,))  # in Celsius
    time.sleep(60) #sleep for a minute

```

XIV. Installation/Setup of Pi.

Use the Raspberry Pi Imager on a different computer connected to your microSD card to put the Debian OS on it. Any version of the Raspberry Pi OS (also known as Raspbian) may work as well, all are Debian-based from the imager [https://www.raspberrypi.com/software/]. The Wi-FI network can be setup during the imaging process, and is important if it is necessary to access the Pi remotely. The computer used to access it must be on the same network.

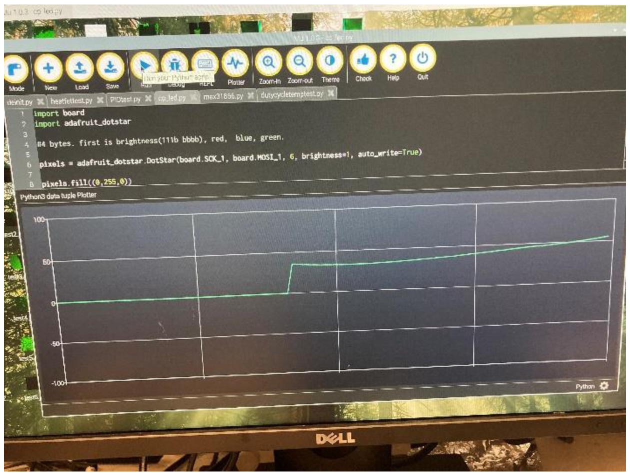
Python should be pre-installed on this. Open the command-line interface (CLI) after connecting the Pi to power and booting it up. The CLI can be accessed remotely [https://www.raspberrypi.com/documentation/computers/remote-access.html] or by simply connecting the Pi to a monitor and keyboard. We needed to connect remotely at first to set up the Bluetooth keyboard/mouse, then did most other operations locally.

Use the following commands in the CLI to get the latest Debian version, and check the python version:

```
sudo apt-get update
sudo apt-get upgrade
python --version
```

Everything stated above to set up an updated Pi, and further steps to install CircuitPython are all included within [16] which is a complete guide.

For editing code on the pi, a combination of two apps were used: the CLI text editor ‘Nano’, and python text editor / Integrated Development Environment (IDE) ‘Mu’ was used. Mu has a handy interface for visually plotting tuple-type output that can be used to assist in PID tuning. On a separate computer, the app notepad++ was used for code text editing and the app FileZilla allowed quick and easy usage of SFTP (SSH File Transfer Protocol) to wirelessly transfer the code and files. The Pi and the separate device simply needed to be connected to the same network.



```
import board
import adafruit_dotstar

#4 bytes. first is brightness(111b bbbb), red,  blue, green.

pixels = adafruit_dotstar.DotStar(board.SCK_1, board.MOSI_1, 6, brightness=1, auto_write=True)

pixels.fill((0,255,0))
```

Figure 17: Photo of Python IDE 'Mu' running on the Raspberry Pi with an HDMI output

Acknowledgment

S. Hasan thanks Michael Golfetto, Natasha Islam, Grey Slatina, Manpreet Singh, Maaz Shaikh, Nick Zuke, and Amar Basu.

References

XV. Bibliography

[1] Super User, “gene-plus - QuantStudio 3D Digital PCR System,” *Gene-plus.com*, 2023. https://www.gene-plus.com/index.php/using-joomla/extensions/templates/atomic/home-page-atomic/88-geneplus-up-date/8-beginners (accessed Mar. 08, 2025).

[2] A. S. Basu, “Droplet Morphometry and Velocimetry,” *Lab on a Chip*, vol. 13, no. 10, 2013, doi: https://doi.org/10.1039/c3lc50074h.

[3] R. P. Ltd, “Raspberry Pi Zero 2 W,” Raspberry Pi Ltd, Apr. 2024. Accessed: Jan. 08, 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf

[4] R. P. Ltd, “Raspberry Pi High Quality Camera,” *Raspberry Pi Ltd*, 2020. https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/ (accessed Dec. 15, 2023).

[5] E. Kodak Co., “KODAK WRATTEN 2 Optical Filter / 12 (Deep Yellow) 0.0,” 2025. Accessed: Sep. 2024. [Online]. Available: https://www.kodak.com/content/products-brochures/Film/Basic-Color-Filters-W2-12.pdf

[6] K. Zhu, “Electrostatic Sensitive Devices SK9822 Specification,” Mar. 2016. Available: https://cdn-shop.adafruit.com/product-files/2343/SK9822_SHIJI.pdf

[7] B. Lin, “Amazon.com : 100X Microscope Lens for Raspberry Pi, C/CS-Mount, 0.12X ~ 1.8X Optical Magnification, Compatible with Raspberry Pi HQ Camera,” *Amazon.com*, 2025. Available: https://www.amazon.com/dp/B0C1CC79TX?language=en-US. [Accessed: Apr. 19, 2025]

[8] "4 PCS Film Heater Plate Adhesive Pad 12V 7W Strip Heater Adhesive Polyimide Heater Plate 25mmx50mm : Industrial & Scientific," *Amazon.com*, 2025. https://www.amazon.com/dp/B07P2RJDPL (accessed Mar. 26, 2025).

[9] OnSemi, "2N3906 General Purpose Transistors PNP Silicon," *Onsemi*, Feb. 2010. https://www.onsemi.com/pdf/datasheet/2n3906-d.pdf (accessed 2024).

[10] I. Rectifier, "IRLB8721PbF HEXFET Power MOSFET," Infineon, 2010. Accessed: 2024. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/irlb8721pbf.pdf

[11] A. Industries, "Adafruit Universal Thermocouple Amplifier MAX31856 Breakout," *www.adafruit.com*, 2024. https://www.adafruit.com/product/3263

[12] L. Ada, "Adafruit MAX31856 Universal Thermocouple Amplifier," *Adafruit*, Jun. 03, 2024. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-max31856-thermocouple-amplifier.pdf (accessed 2024).

[13] Arduino, "Arduino," *Arduino.cc*, 2018. https://www.arduino.cc/ (accessed 2024).

[14] S. Electronics, "Raspberry Pi Zero W," Sparkfun Electronics. Accessed: 2024. [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf

[15] B. J and S. Hymel, "Raspberry Pi SPI and I2C Tutorial," *learn.sparkfun.com*. https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial/all (accessed 2024).

[16] A. Negi, "RPI Zero 2W Board Layout: GPIO Pinout, Specs, Schematic in Detail," *www.etechnophiles.com*, Nov. 17, 2021. https://www.etechnophiles.com/rpi-zero-2w-board-layout-pinout-specs-price/ (accessed 2024).

[17] M. LeBlanc-Williams, "CircuitPython Libraries on Linux and Raspberry Pi," *Adafruit Learning System*, Jun. 30, 2018. https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/overview (accessed 2024).

[18] M. Integrated, "MAX6675 Cold-Junction-Compensated K-Thermocoupleto-Digital Converter," Analog Devices, Jun. 2021. Accessed: 2024. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX6675.pdf

[19] B. Siepert, "Introduction — Adafruit MAX31856 Library 1.0 Documentation," *docs.circuitpython.org*, 2023. https://docs.circuitpython.org/projects/max31856/en/latest/index.html (accessed Mar. 27, 2025).

[20] "American Raspberry Pi Shop," *Pishop.us*, 2023. https://www.pishop.us/ (accessed 2023).

[21] Skoneczny Szymon, "Implementation of PID Controller in Python," *Softinery*, Apr. 13, 2023. https://softinery.com/blog/implementation-of-pid-controller-in-python/ (accessed 2024).

[22] J. Kalsbeek, "PID Control," *pid-tuner.com*, Oct. 02, 2017. https://www.pid-tuner.com/pid-control/ (accessed 2024).

[23] Altium, "CircuitMaker Documentation," *Altium Documentation*, Feb. 10, 2022. https://www.altium.com/documentation/altium-circuitmaker (accessed 2024).

[24] S. Shawcroft, "Adafruit CircuitPython DotStar Library 1.0 documentation," *docs.circuitpython.org*, 2017. https://docs.circuitpython.org/projects/dotstar/en/latest/ (accessed 2024).

[25] K. Rembor, "CircuitPython Essentials," *Adafruit Learning System*, Apr. 02, 2018. https://learn.adafruit.com/circuitpython-essentials/circuitpython-dotstar (accessed 2024).

[26] R. P. Ltd, "RaspiCam Documentation," Raspberry Pi, 2013. Accessed: 2024. [Online]. Available: https://www.raspberrypi.org/app/uploads/2013/07/RaspiCam-Documentation.pdf

[27] N. Mansurov, "Understanding ISO, Shutter Speed and Aperture - A Beginner's Guide," *Photography Life*, Jan. 13, 2010. Available: https://photographylife.com/iso-shutter-speed-and-aperture-for-beginners#what-iso-should-i-set-my-camera-to. [Accessed: 2024]

[28] Python, "Subprocess — Subprocess Management — Python 3.8.5 Documentation," docs.python.org. https://docs.python.org/3/library/subprocess.html (accessed 2024).

[29] pcmanbob, "Re: Raspistill Python Date in Filename," *Raspberry Pi Forums*, 2020. https://forums.raspberrypi.com/viewtopic.php?f=32&t=282623 (accessed 2024).

[30] GeeksforGeeks, "Python Subprocess Module," *GeeksforGeeks*, Dec. 17, 2018. https://www.geeksforgeeks.org/python-subprocess-module/ (accessed 2024).

[31] A. Sedra and K. Smith, *Microelectronic Circuits.* S.L.: Oxford Univ Press Us, 2019.